

Example

To display information in the **calendar** file that pertains to the next two business days:
`calendar`

calendar

A typical **calendar** file might look like this:

```
*/25 - Prepare monthly report
Aug. 12 - Fly to Denver
aug 23 - board meeting
Martha out of town - 8/23, 8/24, 8/25
8/24 - Mail car payment
sat aug/25 - beach trip
August 27 - Meet with Simmons
August 28 - Meet with Wilson
```

If today is Friday, August 24, then the **calendar** command displays:

```
*/25 - Prepare monthly report
Martha out of town - 8/23, 8/24, 8/25
8/24 - Mail car payment
sat aug/25 - beach trip
August 27 - Meet with Simmons
```

Files

\$HOME/calendar

/usr/lib/calprog

/etc/passwd

/tmp/cal*

The program that determines dates.

Used to identify users.

Temporary files.

Related Information

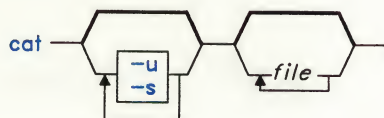
The following commands: “**chmod**” on page 160 and “**mail, Mail**” on page 608.

cat

Purpose

Concatenates or displays files.

Syntax



OL805086

Description

The **cat** command reads each *file* in sequence and writes it to standard output. If you do not specify *file* or specify - (minus) instead of a *file*, **cat** reads from standard input.

Warning: Do not redirect output to one of the input files using the > redirection symbol. If you do this, you will lose the original data in the input file because the shell truncates it before **cat** can read it (see “**sh**” on page 913).

Flags

- s Does not display a message if **cat** cannot find an input file.
- u Does not buffer output.

Examples

1. To display a file at the work station:

```
cat notes
```

This displays the data in the file *notes*. If the file is more than about 23 lines long, some of it will scroll off the screen. To list a file one page at a time, use the **pg** command. (See “**pg**” on page 744 for details.)

2. To concatenate several files:

```
cat section1.1 section1.2 section1.3 >section1
```

This creates a file named `section1` that is a copy of `section1.1` followed by `section1.2` and `section1.3`.

3. To suppress error messages about files that do not exist:

```
cat -s section2.1 section2.2 section2.3 >section2
```

If `section2.1` does not exist, this concatenates `section2.2` and `section2.3`. The result is the same if you do not use the `-s`, except that `cat` displays the error message:

```
cat: cannot open section2.1
```

You may want to suppress this message with the `-s` flag when you use the `cat` command in shell procedures.

4. To append one file to the end of another:

```
cat section1.4 >>section1
```

This appends a copy of `section1.4` to the end of `section1`. The `>>` appends data to the end of `section1`. If you want to replace the file, use the `>`. For more details, see “Redirection of Input and Output” on page 926.

5. To add text to the end of a file:

```
cat >>notes
Get milk on the way home
Ctrl-D
```

Get milk on the way home is added to the end of `notes`. The `cat` command does not prompt; it waits for you to enter text. Press **Ctrl-D** to indicate you are finished.

6. To concatenate several files with text entered from the keyboard:

```
cat section3.1 - section3.3 >section3
```

This concatenates `section3.1`, text from the keyboard, and `section3.3`.

7. To concatenate several files with output from another command:

```
li | cat section4.1 - >section4
```

This copies `section4.1`, and then the output of the `li` command to the file `section4`.

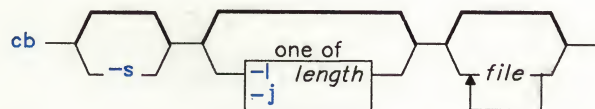
Related Information

The following commands: “**cp**” on page 202, “**pr**” on page 761, and “**sh**” on page 913.

cb

Purpose

Puts C source code into a form that is easily read.

Syntax


OL805170

Description

The **cb** command reads C programs from standard input or from specified *files* and writes them to standard output in a form that shows, through indentations and spacing, the structure of the code. When called without flags, **cb** does not split or join lines. Note that punctuation in preprocessor statements can cause indentation errors.

Flags

- j Joins lines that are split.
- l *length* Splits lines that are longer than *length*.
- s Formats the source code according to the style of Kernighan and Ritchie in *The C Programming Language* (Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1978.).

Example

To create a version of `pgm.c` called `pgm.pretty.c` that is easy to read:

```
cb pgm.c > pgm.pretty.c
```

Related Information

The following command: “**cc**” on page 140.

The discussion of **cb** in *AIX Operating System Programming Tools and Interfaces*.

cc

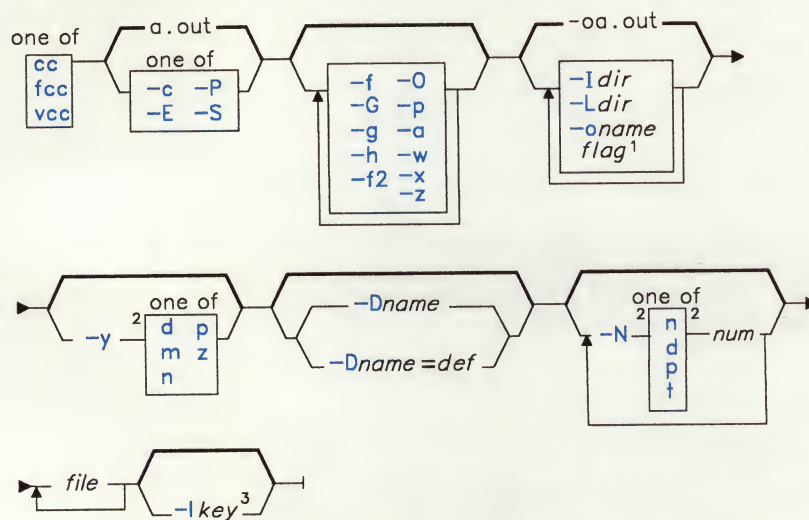
cc

Purpose

Compiles C programs.

Syntax

Ordinary Operation



OL805171

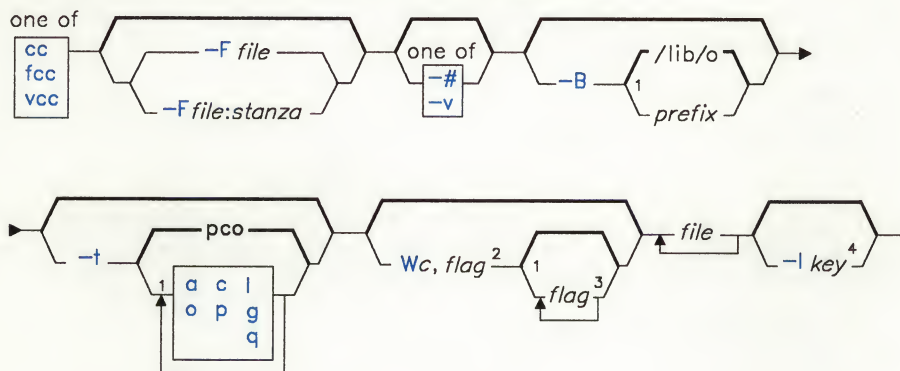
¹ Use any flag belonging to **as**, **cpp**, or **ld** (except **-lkey**).

² Do not put a blank between these items.

³ Put this flag last if used (see the **ld** command).

OL805472

Extended Functions and Debugging



OL805389

¹ Do not put a blank between these items.

² Use any flag belonging to `as`, `cpp`, or `ld` (except `-lkey`).

³ Use any flag from the first diagram (except `-lkey`) or any flag belonging to `as`, `cpp` or `ld`.

⁴ Put this flag last if used (see the `ld` command).

OL805343

Description

The `cc` command runs the C compiler. It accepts files containing C source code, assembler source code, or object code and changes them into a form that the computer system can run. `cc` compiles and assembles source files and then links them with any specified object files, in the order listed on the command line. It puts the resulting executable program in a file named `a.out`.

The `fcc` command is a link to `cc` that compiles programs to run with the Floating-Point Accelerator. `fcc` should only be used on the 032 family of microprocessors. It automatically uses the `-f` flag as well as special versions of the standard libraries that have been compiled for direct floating-point applications. Note that programs compiled with `fcc` can run only on systems that have installed the Floating-Point Accelerator.

The `vcc` command is a link to `cc` that compiles modules to be installed in the VRM. Use the `vrmfmt` command to convert the `a.out` file produced by the `vcc` command to a VRM-compatible object module. The syntax of this command is as follows:

```
vrmfmt infile [outfile]
```

The default output file name is **a.vrm**.

The **cc** command runs the following programs. Each program processes the source file and then sends the results to the next program in the sequence:

cpp The macro preprocessor.

ccom0 The first pass of the compiler.

ccomq The intermediate code optimizer (if you specify the **-O** flag).

This program provides a variety of optimizations to the intermediate code, such as removing loop invariants, eliminating common subexpressions, and allocating registers. The following cannot be optimized:

- Functions that call **setjmp**
- Functions that contain **asm** statements

If you are compiling a large program and the flow optimizer runs out of space, the compiler stops the process and displays a message describing the problem.

ccom1 The second pass of the compiler.

copt The optimizer (if you specify the **-O** flag).

as The assembler.

ld The linkage editor.

You can replace any or all of these passes with your own versions (see the **-B** and **-t** flags). Both **cc** and **fcc** use the **cc.cfg** configuration file, which specifies the standard run time, the link options, and the libraries to be used with each version of the compiler.

Input File Types

The **cc** command recognizes and accepts as input the following file types:

file.c

The name of a C language source file should end with **.c**. After **cc** compiles this source file, it gives the resulting object file the same name, except that it ends in **.o** rather than **.c**. If you use one command both to compile and to load a single C program, the compiler normally deletes the **.o** file when it loads the program. If you use the **-c** flag, the compiler does not delete the **.o** file.

file.i

The name of a file that contains preprocessed C source code ends in **.i**.

file.o

The name of an object file should end in **.o**. The **cc** command sends these files to the **ld** command.

file.s

The name of an assembly language source program should end with **.s**. After **cc** assembles this source file, it gives the resulting object file the same name, except that it ends in **.o** rather than **.s**.

Flags

The **cc** command recognizes several flags. In addition, flags intended to modify the action of the linkage editor (**ld**), the assembler (**as**), or the preprocessor (**cpp**) may also appear on the **cc** command line. **cc** sends any flags it does not recognize to these commands for processing. The following list includes the most commonly used **cpp** flags (**-D**, **-I**), and **ld** flags (**-l**, **-L**, **-o**). See “**as**” on page 61, “**cpp**” on page 210, and “**ld**” on page 557 for a complete list of additional flags.

Note: If you use the **-l** flag, it must be the last entry on the command line, following any file parameters.

Ordinary Operation

- a** Reserves a register for extended addressing. Use this flag if a compiled procedure creates a stack greater than 32,767 bytes. Because this flag causes the compiler to reserve a register for use by the assembler, it reduces the number of available registers by one.
- c** Does not send the completed object file to the **ld** command. With this flag, the output of **cc** is a **.o** file for each **.c** or **.s** file.
- Dname[=def]** Defines *name* as in a **#define** directive. The default *def* is 1.
- E** Runs the named C source file through only the preprocessor and writes the result to standard output.
- f** Generates code that uses the Floating-Point Accelerator or Advanced Floating-Point Accelerator. Programs compiled with this flag will run correctly only on 032 microprocessors configured with either of the Floating-Point Accelerators.
- f2** Generates code that uses the Advanced Floating-Point Accelerator. Programs compiled with this flag will run correctly only on AIX processors configured with the Advanced Floating-Point Accelerator and an Advanced Processor Card.

- g** Produces additional information for use with the **sdb** command (the symbolic debugger).
- G** Indicates that global variables are volatile. The optimizer (**ccomq**) makes fewer transformations when you specify this flag. To make a particular variable volatile, add the "volatile" specification to its declaration.
- h** Treats files with the suffix **.h** in the same way as files with the suffix **.c**.
- I dir** Looks first in *dir*, then looks in the directories on the standard list for **#include** files with names that do not begin with / (slash).
- l[key]** Searches the specified library file, where *key* selects the file **libkey.a**. With no *key*, **-l** selects **libc.a**, the standard system library for C and assembly language programs. **ld** searches for this file in the directory specified by an **-L** flag, then in **/lib** and **/usr/lib**. The **ld** command searches library files in the order in which you list them on the command line.
- L dir** Looks in *dir* for files specified by **-l** keys. If it does not find the file in *dir*, **ld** searches the standard directories.
- N[ndpt] num** Changes the size of the symbol table (**n**), the dimension table (**d**), the constant pool (**p**), or the space for building the parse tree (**t**). Each table must be changed separately. The default size of the symbol table is 1500; the default size of the dimension table is 2000; the default size for the constant pool is 600; the default space for the parse tree is 1000.
- oname** Assigns *name* rather than **a.out** to the output file.
- O** Sends compiler output to the code optimizers.
- p** Prepares the program so that the **prof** command can generate an execution profile. The compiler produces code that counts the number of times each routine is called. If programs are sent to **ld**, the compiler replaces the startup routine with one that calls the **monitor** subroutine at the start (see *AIX Operating System Technical Reference* for a discussion of this subroutine), and writes a **mon.out** file when the program ends normally.
- P** Sends the specified C source file to the macro preprocessor and stores the output in a **.i** file.
- Q!** Controls inlining. The following may be used:
- ?** Shows the reason for not inlining in the output file.
 - name,name . . .** Does not inline *name*.
 - + name,name . . .** Inlines *name*.

- `|num` Limits the size increase of the function in which inlining occurs to *num* intermediate operations. The default *num* is 100.
- `#num` Limits the expansion of an individual call to *num* intermediate operators. The default *num* is 100.
- `-@file` Reads a list of forbidden functions from *file*.
- `+@file` Reads a list of requested functions from *file*.
- Requesting a function to be inlined overrides size constraints.
- S** Compiles the specified C programs, storing assembly language output in a *.s* file.
- w** Prevents printing of warning messages about functions that cannot be optimized.
- X** Produces an assembler listing. This is stored in a file that has the same name as the assembler source file but with the extension *.lst* instead of *.s*.
- y[dmnpz]** Specifies the rounding mode for floating-point constant folding. These modes are specified as follows:
- d** Disables floating-point constant folding.
 - m** Rounds toward negative infinity.
 - n** Rounds to nearest whole number. This is the default action and applies to constant folding in all applicable passes of the compiler.
 - p** Rounds toward positive infinity.
 - z** Rounds toward 0.
- z** Uses the **libm.a** version, or a version specified by the user, of the following transcendental functions:
- | | | | | | |
|------|-------|------|-------|-----|-----|
| acos | asin | atan | atan2 | cos | exp |
| log | log10 | sin | sqrt | tan | |

If this flag is not used, the compiler generates calls to the AIX kernel, or the Advanced Floating Point Accelerator if possible. For more information on **libm.a**, see **math.h** in *AIX Operating System Technical Reference*. For more information on the Advanced Floating Point Accelerator, see **fpfp** in *AIX Operating System Technical Reference*.

Debugging

- Ffile[:stanza]** Uses an alternative *file* and/or *stanza* for **cc** configuration (see *AIX Operating System Technical Reference* for a discussion of the configuration file, **cc.cfg**). If used, this flag must be the first flag on the command line.
- v** Displays the trace as with **-#** and invokes the programs.
- #** Displays a trace of the actions to be taken (for example, invoking the preprocessor), without actually invoking any programs.

Extended Functions

- Bprefix** Constructs path names for substitute preprocessor, compiler, optimizer, assembler, or linkage editor programs. *prefix* defines part of a path name to the new programs. To form the complete path name for each new program, **cc** adds *prefix* to the standard program names (see the discussion of the programs called by **cc** on page 142). For example, if you enter the command:

```
cc testfile.c -B/usr/jim/new
```

cc calls the following compiler programs:

1. /usr/jim/newcpp
2. /usr/jim/newccom0
3. /usr/jim/newccom1
4. /usr/jim/newas
5. /usr/jim/newld

Similarly, if you enter the command:

```
cc testfile.c -B/usr/jim/new/
```

cc calls the following compiler programs:

1. /usr/jim/new/cpp
2. /usr/jim/new/ccom
3. /usr/jim/new/ccom1
4. /usr/jim/new/as
5. /usr/jim/new/ld

The default *prefix* is **/lib/o**.

- t[pcqgoal]** Applies the **-B** flag instructions for constructing file names to only the designated preprocessor (**p**), compiler first (**c**), intermediate code optimizer (**q**), compiler second (**g**), optimizer (**o**), assembler (**a**), or linkage editor (**l**) passes. You can select any combination of **pcqgoal**.

The **-t** flag with no additional **p**, **c**, **q**, **g**, **o**, **a**, or **l** designates by default the preprocessor, compiler and optimizer programs (see the discussion of the programs called by **cc** on page 142).

If you do not specify the **-B** flag when you specify the **-t** flag, the default file name *prefix* is **/lib/n**.

Note: You can specify this *prefix* with the **-B** flag. However, depending on what combination of the **-B** and the **-t** flags you specify, *prefix* can have two possible default values. If you specify **-B** but no accompanying *prefix*, the default *prefix* is **/lib/o**. If you specify the **-t** flag without also specifying the **-B** flag, the default *prefix* is **/lib/n**.

-Wc,flag1[,flag2 . . .]

Gives the listed flags to the compiler program **c**; **c** can be any one of the values [**pcqgoal**] discussed with the **-t** flag. For example, since both **ld** and **as** recognize a **-o** flag, use **-W** to specify the program to which the flag is to be sent. That is, **-Wl,-o** sends it to **ld**. **-Wa,-o** sends it to **as**.

Examples

1. To compile and link a C program, creating an executable **a.out** file:

```
cc pgm.c
```

2. To compile a program, producing an object file to be linked later:

```
cc -c pgm.c
```

This compiles **pgm.c** and produces an object file named **pgm.o**.

3. To compile a program to run on the Floating-Point Accelerator:

```
fcc pgm.c
```

This compiles **pgm.c** using the special libraries **libfc.a** and **libfm.a** instead of the standard libraries **libc.a** and **libm.a**.

4. To view the output of the macro preprocessor:

```
cc -P -C pgm.c
```

This creates a file named **pgm.i** that contains the preprocessed program text including comments. To view this file, use an editor or see “**pg**” on page 744. **cc** passes the **-P** and **-C** flags to the preprocessor. See “**cpp**” on page 210 for more details about them.

5. To predefine macro identifiers:

```
cc -DBUFSIZE=512 -DDEBUG pgm.c
```

This assigns BUFSIZE the value 512 and DEBUG the value 1 before preprocessing. **cc** passes the **-D** flag to the preprocessor.

6. To use **#include** files located in nonstandard directories:

```
cc -I/u/jim/include pgm.c
```

This looks in the directory that contains `pgm.c` for the **#include** files with names enclosed in double quotes (" "), then in `/u/jim/include`, and then in the standard directories. It looks in `/u/jim/include` for **#include** file names enclosed in angle brackets (< >), then in the standard directories. **cc** passes the **-I** flag to the preprocessor.

7. To optimize the object code and produce an assembler listing:

```
cc -S -O pgm.c
```

This uses the optimizing compiler (**-O** is minus, capital oh), and produces an assembler listing in a file named `pgm.s` (**-S**).

Files

<i>file.c</i>	C source file.
<i>file.o</i>	Object file.
<i>file.s</i>	Assembler file.
<i>a.out</i>	Linked output.
<i>/etc/cc.cfg</i>	cc configuration file.
<i>/tmp/ctm*</i>	Temporary.
<i>/lib/cpp</i>	C preprocessor.
<i>/lib/ccom0</i>	Compiler first pass.
<i>/lib/ccomq</i>	Intermediate code optimizer.
<i>/lib/ccom1</i>	Compiler second pass.
<i>/lib/cgen</i>	Compiler.
<i>/lib/copt</i>	Optimizer.
<i>/bin/as</i>	Assembler.
<i>/bin/ld</i>	Linkage editor.
<i>/lib/crt0.o</i>	Run-time startoff.
<i>/lib/mcrt0.o</i>	Run-time startoff for profiling.
<i>/lib/libc.a</i>	Standard library.
<i>/lib/libfc.a</i>	Standard library for use with Floating-Point Accelerator.

/lib/libm.a	Standard math library.
/lib/libfm.a	Standard math library for use with Floating-Point Accelerator.
/lib/librts.a	Runtime services.
/usr/include	Standard directory for #include files.
/usr/tmp/ctm*	Temporary.

Related Information

The following commands: “**as**” on page 61, “**ld**” on page 557, “**cpp**” on page 210, “**prof**” on page 773, and “**sdb**” on page 875.

The discussion of **cc** in *AIX Operating System Programming Tools and Interfaces*, in *C Language Guide and Reference* and in *Assembler Language Reference*.

The **monitor** subroutine, the **a.out** and **cc.cfg** files, the discussion of the Advanced Floating Point Accelerator (**fpfp**), and **math.h** in *AIX Operating System Technical Reference*.

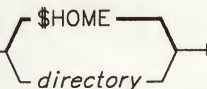
cd

cd

Purpose

Changes the current directory.

Syntax

`cd`  *\$HOME*
directory

OL805087

Description

The **cd** command moves you from your present directory to another. You must have execute (search) permission in the specified *directory*.

If you do not specify a *directory*, **cd** moves you to your login directory (**\$HOME**). If the specified *directory* name is a full path name, it becomes the current directory. A full path name begins with a / (slash—root directory), with a . (dot—current directory), or with a . . (dot dot—parent directory). If the directory name is not a full path name, **cd** searches for it relative to one of the paths specified by the **\$CDPATH** shell variable. This variable has the same syntax as, and similar semantics to, the **\$PATH** shell variable. (See “Shell Variables and Command-Line Substitutions” on page 917 for a discussion of these variables.)

Examples

1. To change to your home directory:
`cd`
2. To change to an arbitrary directory:
`cd /usr/include`

This changes the current directory to `/usr/include`. Now file path names that do not begin with `/` or `../` specify files located in `/usr/include`.

3. To go down one level of the directory tree:

```
cd sys
```

If the current directory is `/usr/include` and if it contains a subdirectory named `sys`, then `/usr/include/sys` becomes the current directory.

4. To go up one level of the directory tree:

```
cd ..
```

The special file name `..` (dot-dot) refers to the directory immediately above the current directory. However, under symbolic links, `..` (dot-dot) refers to the parent directory of the symbolic link, not to the directory above the current directory.

Related Information

The following commands: “**cs**h” on page 225, “**pwd**” on page 800, and “**sh**” on page 913.

Note: The **cs**h and **sh** commands each contain a built-in subcommand named **cd**. The description of **cd** given above applies only to **sh**. For more information see the **cs**h command.

The **chdir** system call in *AIX Operating System Technical Reference*.

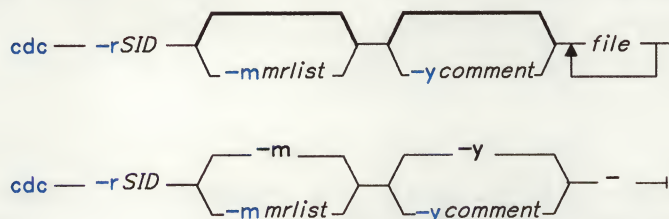
cdc

cdc

Purpose

Changes the comments in a Source Code Control System (SCCS) delta.

Syntax



OL805088

Description

The **cdc** command changes the **Modification Requests** (MRs) and comments for the **SID** specified by the **-r** flag for each named **Source Code Control System** (SCCS) *file*. If you specify a directory name, **cdc** performs the requested actions on all SCCS files in that directory (that is, all files with names that have the **s.** prefix). If you specify a **-** (minus) in place of *file*, **cdc** reads standard input and interprets each line as the name of an SCCS file. For more information on SCCS comments and Modification Requests, see *AIX Operating System Programming Tools and Interfaces*.

You can change the comments and MRs for an SID only if you made the SID or you own the file and the directory. For more information on the permissions needed to change SCCS files, see "SCCS Files" on page 478.

Flags

-m[mrlist] Supplies a list of MR numbers for **cdc** to add or delete in the SID specified by the **-r** flag. You can only use this flag if the *file* has the **v** header flag set (see Figure 1 on page 44). A null MR list has no effect.

In the *mrlist*, MRs are separated by blanks, tab characters, or both. To delete an MR, precede the MR number with an ! (exclamation point). If the MR you want to delete is currently in the list of MRs, it is changed into a comment line. **cdc** places a list of all deleted MRs in the comment section of the delta and precedes them with a comment line indicating that the following MRs were deleted.

If you do not specify the **-m** flag, and the **v** header flag is set, MRs are read from standard input. If standard input is a work station, **cdc** prompts you for the MRs. The first new-line character not preceded by a backslash ends the list on the command line. **cdc** continues to take input until it reads an end-of-file character (**Ctrl-D**) or a blank line. MRs are always read before comments (see the **-y** flag).

If the **v** flag has a value, **cdc** interprets the value as the name of a program which validates the MR numbers. If the MR number validation program returns a nonzero exit value, **cdc** stops and does not change the MRs.

- rSID** Specifies the SCCS identification number of the delta for which **cdc** will change the comments or MRs.
 - y[comment]** Specifies text to replace any *comment* already existing for the delta specified by the **-r** flag. **cdc** keeps the existing comments and precedes them by a comment line stating that they were changed. A null *comment* has no effect.
- If you do not specify **-y**, **cdc** reads comments from standard input until it reads an end-of-file character. If the standard input is a work station, **cdc** prompts for the comments and also allows a blank line to end input. If the last character of a line is a \ (backslash), **cdc** ignores it and continues to read standard input.

Note: If **cdc** reads standard input for file names (that is, when you specify a file name of -), you must use the **-y** and **-m** flags.

Related Information

The following commands: “**admin**” on page 41, “**delta**” on page 310, “**get**” on page 477, “**help**” on page 513, and “**prs**” on page 781.

The **sccsfile** file in *AIX Operating System Technical Reference*.

The discussion of SCCS in *AIX Operating System Programming Tools and Interfaces*.

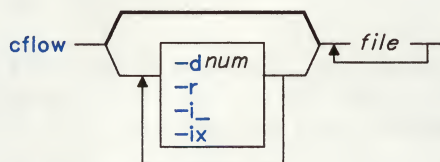
cflow

cflow

Purpose

Generates a C flow graph of external references.

Syntax



OL805172

Description

The **cflow** command analyzes C, **yacc**, **lex**, assembler, and object *files* and writes a chart of their external references to standard output.

It sends files with suffixes **.y**, **.l**, and **.c** to the **yacc**, **lex**, and **cpp** commands for the appropriate processing. This step is bypassed for **.i** files. It then runs the output of this processing through the first pass of **lint**. It assembles files which end in **.s**, extracting information from the symbol table (as it does with **.o** files). From this output, **cflow** produces a graph of external references, which it writes to standard output.

Each line of output begins with a line number followed by sufficient tabs to indicate the level of nesting. Then comes the name of the global, a colon, and its definition. This name is normally a function not defined as external and not beginning with an underline character; see the **-i_** inclusion flag on p. 155. For information extracted from C source files, the definition consists of an abstract type declaration (for example, **char***), the name of the source file, surrounded by angle brackets, and the line number on which the definition was found. Definitions extracted from object files contain the file name and location counter under which the symbol appeared. **cflow** deletes leading underline characters in C-style external names.

Once **cflow** displays the definition of a name, later references to it contain only the **cflow** line number where the definition may be found. For undefined references, **cflow** displays only **<>** (redirection symbols).

Related Information

The “Overview of International Character Support” in *Managing the AIX Operating System*.

The discussion of Japanese Language Support in *Japanese Language Support User's Guide*.


calendar

calendar

Purpose

Writes reminder messages to standard output.

Syntax

calendar 

OL805169

Description

The **calendar** command reads a file named **calendar**, which you create in your current (usually home) directory. It writes to standard output any line in the file that contains today's or tomorrow's date.

The **calendar** command recognizes date formats such as Dec. 7 or 12/7. It also recognizes the special character * (asterisk). It interprets */7, for example, as signifying the seventh day of every month. **calendar** does not recognize formats such as 7 December, 7/12, or DEC. 7.

On Fridays, **calendar** writes all lines containing the dates for Friday, Saturday, Sunday, and Monday. It does not, however, recognize holidays, so "tomorrow" is the holiday rather than the next working day.

For you to get reminder service, your **calendar** should have read permission for others (see "**chmod**" on page 160).

Flag

- Calls **calendar** for everyone having a file **calendar** in his home directory and sends any reminders by mail.

If the nesting level becomes too deep to display in available space, pipe the output from **cflow** to the **pr** command, using the **-e** flag to compress the tab expansion to something less than every eight spaces.

Note: Files produced by **lex** and **yacc** cause the reordering of line number declarations which can confuse **cflow**. To get proper results, feed **cflow** the **yacc** or **lex** input.

Flags

In addition to the following, **cflow** recognizes the **-I**, **-D**, and **-U** flags of the **cpp** command.

- dnum** Sets to decimal integer *num* the depth at which the flow graph is cut off. By default this is a very large number. Do not set the cutoff depth to a nonpositive integer.
- ix** Includes external and static data symbols. The default includes only functions.
- i_** Includes names that begin with an underline character. The default excludes these functions (and corresponding data if **-ix** is used).
- r** Produces an inverted listing which shows the callers of each function, sorted by called function.

Related Information

The following commands: “**as**” on page 61, “**cc**” on page 140, “**lex**” on page 562, “**lint**” on page 577, “**nm**” on page 705, “**pr**” on page 761, and “**yacc**” on page 1237.

The discussion of **cflow** in *AIX Operating System Programming Tools and Interfaces*.

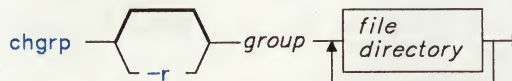
chgrp

chgrp

Purpose

Changes the group ownership of a file or directory.

Syntax



OL805090

Description

The **chgrp** command changes the group associated with the specified *file* or *directory* to *groupname* or *groupID*. If you do not own the file, you must have superuser authority to change the group ID.

If the file or directory resides in a Distributed Services remote virtual file system, the translated group ID is used.

Flag

-r Causes the untranslated group ID to be used. Applies only to files or directories that reside in a Distributed Services remote virtual file system.

Examples

To change the group ownership of the file or directory named `proposals` to `staff`:

```
chgrp staff proposals
```

The group access permissions for `proposals` now apply to the `staff` group.

Files

`/etc/group` File that identifies all known groups.

Related Information

The following command: **"groups"** on page 506.

The **chown** and **chownx** system calls and the **group** file in *AIX Operating System Technical Reference*.

"Distributed Services **id** Translation" in *Managing the AIX Operating System*.

chkcomp

chkcomp

Purpose

Checks compatibility between a code server and an active-service client.

Syntax

```
chkcomp — -n node — -w directory —
```

AJ2FL261

Description

The **chkcomp** command checks compatibility between a code server and an active-service client before allowing active service. Generally, this command is run by the **chnstate** command. You must be a member of the system group or operating with superuser authority to run this command.

When the command runs, it compares one or more programs installed at the code server to a program or program subset installed at the client and identifies incompatibilities between version, release, or level. If the client and server are compatible, **chkcomp** ends. If the client and server are not compatible, **chkcomp** writes a comprehensive set of install and update orders to a **cs.compat** file.

There are two types of incompatibilities: incompatibilities that can be fixed by the system and incompatibilities that require manual intervention. These types of incompatibilities are handled in the following manner:

- If the incompatibilities can be fixed by the system and the upgrade mode is automatic, the system automatically initiates the required install and update orders.
- If the incompatibilities can be fixed by the system and the upgrade mode is manual, the system displays the list of required manual actions.
- If the incompatibilities cannot be fixed by the system, the client defaults to stand-alone mode and the user is responsible for initiating the required install and update actions (orders in the **cs.compat** file) to resolve the incompatibilities.

The upgrade mode is set by a value in the **serverattach** file or by **-m** flag to the **chnstate** command. For more information on the compatibility rules and which types of incompatibilities can be fixed by the system, see *Managing the AIX Operating System*.

Flags

- n node** Specifies the node ID or nickname of the code server. This flag and *node* are required and must be specified.
- w directory** Specifies the directory on the client where the code server's root file system is mounted. This mount must be done before **chkcomp** is run. If **chkcomp** is run from the **chgstate** command, the mount is done for you. If you want to run **chkcomp** directly, you must run the **mount** command before you run the **chkcomp** command. This flag and directory are required and must be specified.

Example

To create and mount a directory where a code server's root file system can reside on your system and to check compatibility between the code server and your system:

```
mkdir /tmp/nick
mount -n darlene / /tmp/nick
chkcomp -n darlene -w /tmp/nick
```

This makes the directory /tmp/nick, mounts the / (root) directory of the code server darlene on this directory, and then checks the compatibility of the code server and your system.

Files

/etc/codeserve/cs.compat	Contains information on client and server compatibility.
/etc/codeserve/serverattach	Contains code service attribute information.
/usr/lpp/lpp-name/lpp.hist	Contains history files for both the code server and the client
/var/lpp/lpp-name/lpp.hist	Contains history files for both the code server and the client systems.

Related Information

The following command: "**chgstate**" on page 164.

The **/etc/codeserve/servattach** file in *AIX Operating System Technical Reference*.

The discussion of code service and the **/etc/codeserve/cs.compat** file in *Managing the AIX Operating System*.

chmod

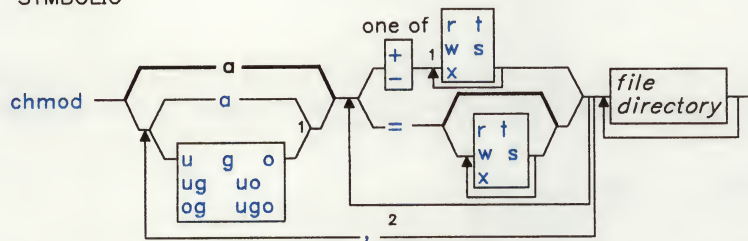
chmod

Purpose

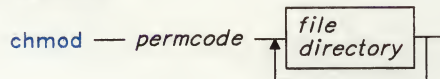
Changes permission codes.

Syntax

SYMBOLIC



ABSOLUTE



¹ Do not put a blank between these items.

² Do not put a blank on either side of the comma.

OL805091

Description

The **chmod** command modifies the read, write, execute (*file*), or search (*directory*) permission codes of specified files or directories. You can use either symbolic or absolute mode to specify the desired permission settings.

You can change the permission code of a file or directory only if you own it or if you are operating with superuser authority.

Symbolic Mode

When you use the symbolic mode to specify permission codes, the first set of flags selects the permission field, as follows:

- u** User (owner)
- g** Group
- o** All others
- a** User, group, and all others (same effect as **ugo**). This is the default permission field.

The second set of flags selects whether permissions are to be taken away, added, or set exactly as specified:

- Removes specified permissions
- +** Adds specified permissions
- =** Clears the selected permission field and sets it to the code specified. If you do not specify a permission code following **=**, **chmod** removes all permissions from the selected field.

The third set of flags of the **chmod** command selects the permissions as follows:

- r** Read permission.
- w** Write permission.
- x** Execute permission for files; search permission for directories.
- s** Set User-ID or Set Group-ID permission. This permission bit sets the effective user-ID or group-ID to that of the *file* whenever the *file* is run. Use this permission setting in combination with the **u** or **g** field to allow temporary or restricted access to files not normally accessible to other users. An **s** appears in the user or group execute position of a long listing (see “ls” on page 595 or “li” on page 567), to show that the file runs Set User-ID or Set Group-ID.
- t** The save text permission. Setting this permission bit causes the text segment of a program to remain in virtual memory after its first use. The system thus avoids having to transfer the program code of frequently-accessed programs into the paging area. A character special file with this bit set is a multiplexed file. You can specify this permission only with the **u** field. A **t** appears in the execute position of the All Others field to indicate that the file has this bit (the *sticky* bit) set.

You can specify multiple symbolic modes, separated with commas. Do not separate items in this list with spaces. Operations are performed in the order they appear from left to right.

Absolute Mode

The **chmod** command also permits you to use octal notation to set each bit in the permission code. **chmod** sets the permissions to the *permcode* you provide. This *permcode* is constructed by combining (the logical OR of) the following values:

4000	Sets user-ID on execution
2000	Sets group-ID on execution
1000	Retains memory image after execution (executable file)
1000	Indicates multiplexed character special file
0400	Permits read by owner
0200	Permits write by owner
0100	Permits execute or search by owner
0040	Permits read by group
0020	Permits write by group
0010	Permits execute or search by group
0004	Permits read by others
0002	Permits write by others
0001	Permits execute or search by others

All permission bits not explicitly specified are cleared.

Examples

1. To add a type of permission to several files:

```
chmod g+w chap1 chap2
```

This adds write permission for group members to the files `chap1` and `chap2`.

2. To make several permission changes at once:

```
chmod go-w+x mydir
```

This denies group members and others the permission to create or delete files in `mydir` (`go-w`). It allows them to search `mydir` or use it in a path name (`go+x`). This is equivalent to the command sequence:

```
chmod g-w mydir
chmod o-w mydir
chmod g+x mydir
chmod o+x mydir
```

3. To permit only the owner to use a shell procedure as a command:

```
chmod u=rwx,go= cmd
```

This gives read, write, and execute permission to the user who owns the file (`u=rwx`). It also denies the group and others the permission to access `cmd` in any way (`go=`).

If you have permission to execute the shell command file `cmd`, then you can run it by entering:

```
cmd
```

This may not work in some cases, depending on the value of the shell variable **PATH**. See page 923 for more information about **PATH**.

4. To use Set-ID modes:

```
chmod ug+s cmd
```

When `cmd` is executed, this causes the effective user and group IDs to be set to those that own the file `cmd`. Only the effective IDs associated with the subprocess that runs `cmd` are changed. The effective IDs of the shell session remain unchanged.

This feature allows you to permit restricted access to important files. Suppose that the file `cmd` has the Set-User-ID mode enabled and is owned by a user called `dbms`. `dbms` is not actually a person, but might be associated with a database management system. The user `betty` does not have permission to access any of `dbms`'s data files. However, she does have permission to execute `cmd`. When she does so, her effective user ID is temporarily changed to `dbms`, so that the `cmd` program can access the data files owned by `dbms`.

This way `betty` can use `cmd` to access the data files, but she cannot accidentally damage them with the standard shell commands.

5. To use the absolute mode form of the **chmod** command:

```
chmod 644 text
```

This sets read and write permission for the owner, and it sets read-only mode for the group and others.

Related Information

The following commands: “**ls**” on page 595, “**li**” on page 567, and “**umask**” on page 1110.

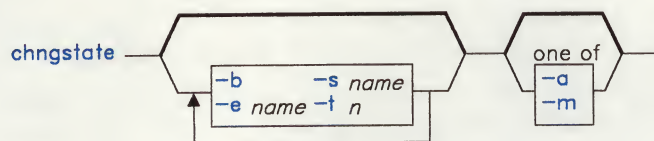
chgstate

chgstate

Purpose

Changes the state of a code service client to either active-service or stand-alone.

Syntax



AJ2FL263

Description

Use the **chgstate** command to change the state of a code service client to active-service or stand-alone. You must be a member of the system group or operating with superuser authority to run this command. It is also run by the **rc** command during system initialization.

When the command runs, it validates and processes the code service attribute file **/etc/codeserve/serverattach**. The contents of this file tell **chgstate** whether the system should be an active-service system or a stand-alone system.

If the target state is stand-alone, **chgstate** runs **rc.standalone** and **rc.include**.

If the target state is active-service, **chgstate** checks the attribute file to determine:

- The server to use
- The time between attach attempts for the specified server
- The maximum time to attempt individual server attach
- The upgrade mode (automatic or manual).

Then **chgstate** runs the **chkcomp** command to check for client-server compatibility before attempting to attach to the code server in active-service mode.

If the client and server are compatible, **chgstate** runs **rc.actvsrve** and **rc.include** to attach the client to the code server in active-service mode. If the client and server are not compatible, **chkcomp** writes a comprehensive set of install and update orders to a **cs.compat** file.

There are two types of incompatibilities: incompatibilities that can be fixed by the system and incompatibilities that require manual intervention. These incompatibilities are handled in the following manner:

- If the incompatibilities can be fixed by the system and the upgrade mode is automatic, the system attempts to fix incompatibilities by calling the internal command **installc** to upgrade the install and update state of the client.
- If the incompatibilities can be fixed by the system and the upgrade mode is manual, the system displays the list of required manual actions.
- If the incompatibilities cannot be fixed by the system, the client defaults to stand-alone mode.

In the latter two cases, the user is responsible for initiating the required install and update actions (orders in the **cs.compat** file) to resolve the incompatibilities. The system will not attach a client as long as incompatibilities exist. To manually resolve incompatibilities:

- Run **installp** or **updatep** to install or update programs on the client.
- Rerun the **chgstate** command to verify compatibility and attach to the server as an active-service client.

For more information on the compatibility rules and which types of incompatibilities can be fixed by the system, see *Managing the AIX Operating System*.

Notes:

1. The upgrade mode parameter (automatic or manual) must be specified in the attribute file or an error will result. However, you can override the attribute in the file by specifying the **-a** or **-m** flag in the **chgstate** command.
2. If **chgstate** encounters a server timeout error while attempting to attach to a server, it reads the next system attribute file stanza and attempts to attach to that server according to the stanza attributes. Any other error causes the system to come up in stand-alone mode.
3. You can also run the **ckcomp** command from the command line to get a list of which programs are incompatible. However, **ckcomp** does not leave you attached to the server in active-service mode.

Flags

- a** Uses automatic upgrade mode when attempting to attach to any server with a target state of active-service. This flag overrides the mode set in the system attribute file stanzas.
- b** Prevents **chgstate** from running a **killall** command. This flag should be used only when **chgstate** is run from **/etc/rc** during a system boot.
- e name** Excludes the attribute file stanza specified by *name*. To exclude more than one stanza, enter the **-e** flag for each stanza.

chgstate

- m** Uses manual upgrade mode when attempting to attach to any server with a target state of active-service. This flag overrides the mode set in the system attribute file stanzas.
- s name** Starts processing with this system attribute file stanza.
- t n** Specifies the total time in seconds to attempt to attach to any server. The time, *n*, must be greater than or equal to 60.

Internal Commands

The **chgstate** command uses the following internal commands. Because they are internal commands, they do minimum validation of input parameters.

installc

The **installc** command installs a full program or subset program. It uses the following syntax:

```
installc -r path /etc/codeserve/cs.compat file
```

AJ2FL140

The **installc** command attempts to upgrade the installation or update state of an active-service client to make it fully compatible with a specific server by installing a full program or a subset program on the active-service client. Generally, **installc** is run by the **chgstate** command when there is an incompatibility that can be automatically corrected by installing a complete or subset program. A user's path would not normally include this command. It is located at **/etc/codeserve/installc**. The **installc** command requires you to be a member of the system group or operating with superuser authority.

When run, **installc** processes install and update requests from an input file in the format defined in the file **cs.compat**. Install requests result in a call to the command **installp** and update requests result in a call to the internal command **updatec**.

The **chgstate** command runs this command when the **cs.compat** file contains only install, update, or install and update records. If **cs.compat** does not exist or contains dbos, uplevel, or unknown records, **installc** will not be run. In this case the user must manually upgrade the client by running the **installp** or **updatep** command for the appropriate programs.

If install records exist, then **chgstate** mounts the server directory **/usr/lpp.install** prior to running **installc**. If update records exist, then **chgstate** mounts the server directory **/usr/lpp.update**.

The **-r path** parameter is used to pass the path that **installc** must use to gain access to the server's **/usr/lpp.install** directory. It represents a local path where **chgstate** has mounted the **/** (root) directory of the server.

The **file** is an input file in **cs.compat** format. This file contains specific install and update requests required to make the client fully compatible with a specific code server.

updatec

The **updatec** command controls the update process for complete or subset programs on active-service clients in a code service environment. It uses the following syntax:

```
updatec — -r path — /etc/codeserve/cs.compat — file —
```

AJ2FL141

Generally, this command is run from the **installc** internal command to upgrade the update state of an active service client and make it fully compatible with a specific server on an active service network. A user's path would not normally include this file. It is located at **/etc/codeserve/updatec**. You must be a member of the system group or be operating with superuser authority to run this command.

When run, this command processes update requests from an input file in the format defined in the file **cs.compat**. Update requests result in a call to the **updatep** internal command, **inuupdt**.

This command is run by **installc** only when the **cs.compat** file contains update records. If **cs.compat** does not exist or contains dbos, uplevel, or unknown records, **installc** will not be called.

The **-r path** parameter is used to pass the path that **updatec** must use to gain access to the server's **/usr/lpp.update** directory. It represents a local path where **chgstate** has mounted the **/** (root) directory of the server.

The **file** is an input file in **cs.compat** format. The file contains specific update requests required to make this client fully compatible with a specific code server.

Files

/etc/codeserve/cs.compat	Contains information on client and server compatibility.
/etc/codeserve/serverattach	Contains code service attribute information.
/etc/rc	Performs normal startup initialization.
/etc/rc.standalone	Initializes stand-alone system.

chgstate

/etc/rc.actvsrv
/usr/lpp.install

Initializes active-service client.

Server directory containing backup format files required to do installations.

/usr/lpp.update

Server directory containing backup format files required to do updates.

Related Information

The following commands: “**chkcomp**” on page 158, “**installp**” on page 529, “**rc**” on page 806, “**updatep**” on page 1122.

The `/etc/codeserve/serverattach` file in *AIX Operating System Technical Reference*.

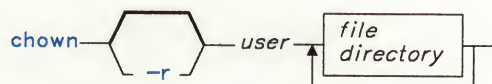
The discussion of code service and the `/etc/codeserve/cs.compat` file in *Managing the AIX Operating System*.

chown

Purpose

Changes the owner of files or directories.

Syntax



OL805095

Description

The **chown** command changes the owner of the specified *files* or *directories* to *username* or *userID*. The group associated with the file or directory is not affected.

Note: If you give ownership of a file or directory to another user, you cannot regain ownership unless you have superuser authority.

If the file or directory resides in a Distributed Services remote virtual file system, the translated user ID is used.

Flag

-r Causes the untranslated user ID to be used. Applies only to files or directories that reside in a Distributed Services remote virtual file system.

Example

```
chown jim program.c
```

The user access permissions for `program.c` now apply to `jim`. As the owner, `jim` can use **chmod** to permit or deny the other users access to `program.c`. See “**chmod**” on page 160 for details.

Files

`/etc/passwd` File that contains user IDs.

chown

Related Information

The following command: “**passwd**” on page 735.

The **chown** and **chownx** system calls and the **passwd** file in *AIX Operating System Technical Reference*.

“Distributed Services **id** Translation” in *Managing the AIX Operating System*.

chparm

Purpose

Changes or examines system parameters.

Syntax

```
chparm nodename kernel-image
chparm nodename=newvalue kernel-image
```

OL805093

Description

The **chparm** command lets you change a system parameter or look at its current setting. Currently, only the **nodename** parameter may be examined or changed. The name assigned cannot be longer than eight characters. If you do not assign a *newvalue*, **chparm** writes the current value of **nodename** to standard output. The default *kernel-image* is **/unix**.

Changes do not affect the running system. You must restart the system for the change to become effective.

Examples

1. To display the **nodename** of your system:

```
chparm nodename
```

This displays the **nodename** of **/unix**, which is a file containing the kernel of the AIX Operating System. This file is loaded and run when you start up the computer.

2. To change the **nodename** of a system:

```
chparm nodename=COMP-CTR /unix.compctr
```

This changes the **nodename** of **/unix.compctr** to **COMP-CTR**. **/unix.compctr** is a file that contains an alternate version of the operating system kernel. The change does not affect the running system, even if you change the **/unix** kernel.

chroot

chroot

Purpose

Changes the root directory of a command.

Syntax

`chroot` — *directory* — *command* —

OL805094

Description

Warning: If special files in the new root have different major and minor device numbers than they have in the real root, it is possible to overwrite the file system.

The **chroot** command can be used only by a user operating with superuser authority (see “su” on page 1026). If you have superuser authority, the **chroot** command changes the root directory to the specified *directory* when executing *command*. The first / (slash) in any path name changes to *directory* for the specified *command* and any of its children.

Notice that:

chroot *directory* *command* > *file*

creates the *file*. relative to the original root, not the new one.

The *directory* path name is always relative to the current root. Even if a **chroot** is in effect, *directory* is relative to the current root of the running process.

Several programs may not operate properly after **chroot** has been run. For example, the command `ls -l` will fail to give user and group names if the current root location makes `/etc/passwd` beyond reach. In addition, utilities that depend on description files produced by the **ctab** command (see page 257) may fail altogether if these files are also not in the new root file system. It is your responsibility to ensure that all vital data files are present in the new root file system and that the path names accessing such files are changed as necessary.

Examples

1. To run a subshell with another file system as the root:

```
chroot /diskette0 /bin/sh
```

This makes the directory name `/` refer to **/diskette0** for the duration of the command **/bin/sh**. It also makes the original root file system inaccessible. The file system on **/diskette0** must contain the standard directories of a root file system. In particular, the shell will look for commands in **/bin** and **/usr/bin** on the **/diskette0** file system.

Running the command **/bin/sh** creates a subshell, which runs as a separate process from your original shell. Press END OF FILE (**Ctrl-D**) to end the subshell and go back to where you were in the original shell. This restores the environment of the original shell, including the meanings of the current directory (**.**) and the root directory (**/**).

2. To run a command in another root file system and save the output:

```
chroot /diskette0 /bin/cc -E /u/bob/prog.c >prep.out
```

This runs the **/bin/cc** command with `/` referring to **/diskette0**. It saves the output in the file `prep.out`, which is in the original root file system.

This runs the C language preprocessor (**/bin/cc -E**) on the file `/diskette0/u/bob/prog.c`, reading **#include** files from **/diskette0/usr/include**, and putting the preprocessed text in `prep.out` on the primary root file system.

Related Information

The following commands: “**cc**” on page 140, “**cpp**” on page 210, and “**sh**” on page 913.

The **chdir** and **chroot** system calls in *AIX Operating System Technical Reference*.

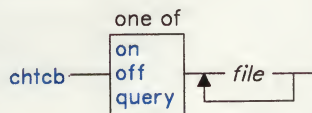
chtcb

chtcb

Purpose

Sets or queries the tcb attribute of a file.

Syntax



A5AC5021

Description

The **chtcb** command sets or queries the **tcb** attribute of the specified files. The **tcb** attribute of a file should be on in that file is in the trusted computing base; otherwise, is should be off.

A file must be in the trusted computing base if it is to be executed from the trusted shell (**tsh**).

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Related Information

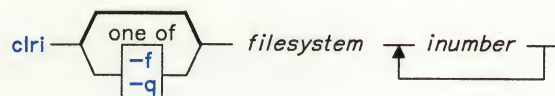
The following commands: “**sysck**” on page 1031 and “**tsh**” on page 1100.

clri

Purpose

Clears the specified i-node.

Syntax



OL805097

Description

Warning: Use this command only in emergencies and with extreme care.

The **clri** command is used to clear i-node entries for files that do not appear in a directory. In general, you do not need to use this program because **fsck** can deal with most file system inconsistencies.

Always run **fsck** on a file system after you have used **clri** on it, because it may create dangling directory references or missing blocks. These can be fixed if they are attended to promptly. Do not run the system when the file system has dangling directory references or a bad free list.

The **clri** command zeros over the flags' word of the i-node, thus freeing it for reallocation. The *inumber* parameter specifies the i-node and *filesystem* specifies the file system it is on. *inumber* should be a decimal number, while *filesystem* can be either the name of the device on which the file system resides or the name by which it is normally mounted.

If you use **clri** to remove an i-node that does appear in a directory, you should track down and remove all of these entries. Otherwise, when the i-node is reallocated to some new file, the old entry will still point to that file. At that point removing the old entry destroys the new file and the new entry again points to an unallocated i-node.

By default, the **clri** command displays some information about the file and asks for confirmation before it destroys the file. If you enter a y or yes, the file is destroyed.

Since **clri** only zeros the flags' word of the i-node, if you destroy the wrong file, you can recover the file by using the **fsdb** command to restore the flags' word.

Note: If the file is open, **clri** is likely to be ineffective. For this reason, you should run **clri** only on an unmounted file system.

clri

Flags

- f Destroys the file without confirmation, but writes a description of the file.
- q Destroys the file without confirming or writing a description of the file.

Example

To clear i-nodes 170 and 368 of the file system **/diskette0** and then clean up the file system:

```
clri /diskette0 170 368  
fsck /diskette0
```

Related Information

The following commands: “**fsck**, **dfscck**” on page 445 and “**fsdb**” on page 450.

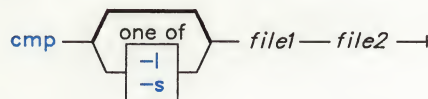
The **fs** file in *AIX Operating System Technical Reference*.

cmp

Purpose

Compares two files.

Syntax



OL805157

Description

The **cmp** command compares *file1* and *file2* and writes the results to standard output. If you specify a - (minus) for *file1*, **cmp** reads standard input. Under default conditions, **cmp** displays nothing if the files are the same. If they differ, **cmp** displays the byte and line number at which the first difference occurs. If one file is an initial subsequence of the other (that is, if **cmp** reads an end-of-file character in one file before finding any differences), **cmp** notes this. Normally, you use **cmp** to compare non-text files and the **diff** command to compare text files.

Flags

- l Displays, for each difference, the byte number in decimal and the differing bytes in octal.
- s Returns only an exit value. (0 indicates identical files; 1 indicates different files; 2 indicates inaccessible file or a missing argument)

Examples

1. To determine whether two files are identical:

```
cmp prog.o.bak prog.o
```

This compares *prog.o.bak* and *prog.o*. If the files are identical, then a message is not displayed. If the files differ, then the location of the first difference is displayed.

cmp

For instance:

```
prog.o.bak prog.o differ: char 5, line 1
```

If the message `cmp: EOF on prog.o.bak` is displayed, then the first part of `prog.o` is identical to `prog.o.bak`, but there is additional data in `prog.o`.

2. To display each pair of bytes that differ:

```
cmp -l prog.o.bak prog.o
```

This compares the files, and then displays the byte number (in decimal) and the differing bytes (in octal) for each difference. For example, if the fifth byte is octal 101 in `prog.o.bak` and 141 in `prog.o`, then `cmp` displays:

```
5 101 141
```

3. To compare two files without writing any messages:

```
cmp -s prog.c.bak prog.c
```

This gives an exit value of 0 if the files are identical, 1 if different, or 2 if an error occurs. This form of the command is normally used in shell procedures. For example:

```
if cmp -s prog.c.bak prog.c
then
    echo No change
fi
```

This partial shell procedure displays `No change` if the two files are identical. See page 930 for details about the `if` command.

Related Information

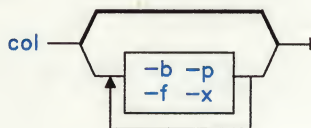
The following commands: “**comm**” on page 183, “**diff**” on page 320, and “**sh**” on page 913.

col

Purpose

Processes text having reverse linefeeds and forward/reverse half-linefeeds for output to standard output.

Syntax



OL805173

Description

The **col** command reads from standard input and writes to standard output. It performs the line overlays implied by reverse line feeds (ASCII ESC-7), and by forward and reverse half-line feeds (ASCII ESC-9 and ASCII ESC-8). **col** is particularly useful for filtering multicolumn output made by the **nroff .rt** command and output from the **tbl** command. The input format accepted by **col** matches the output format produced by **nroff -T37** or by **nroff -Tlp**. Use **-T37** and the **col -f** flag if the output is being sent to a device that can interpret half-line motions; use **-Tlp** otherwise.

The **col** command assumes that the ASCII control characters SO (\017) and SI (\016) begin and end text in an alternate character set. **col** remembers the character set each input character belongs to, and on output generates SI and SO characters as appropriate to ensure that each character is printed in the correct character set.

On input, **col** accepts only the control characters for Space, Backspace, Tab, Return, the new-line character, SI, SO, VT, and ESC-7, 8, or 9. VT (\013) is an alternate form of full reverse line feed included for compatibility with some earlier programs of this type. **col** ignores all other non-printing characters.

Notes:

1. The maximum number of lines that can be backed up is 128.
2. Up to 800 characters, including backspaces, are allowed on a line.
3. Local vertical motions that would result in backing up over the first line are ignored. As a result, the first line must not contain any superscripts.

Flags

- b Assumes that the output device in use is not capable of backspacing. In this case, if two or more characters are to appear in the same position, only the last one read appears in the output.
- f Suppresses the default treatment of half-line motions in the input. Normally, **col** does not emit half-line motions on output, although it does accept them in its input. With this flag, output may contain forward half-line feeds (ESC-9) but not reverse line feeds (ESC-7 or ESC-8).
- p Displays unknown escape sequences as characters, subject to overprinting from reverse line motions. Normally, **col** ignores them. You should be fully aware of the textual position of escape sequences before you use this flag.
- x Suppresses changing the white space to tabs. Without this flag, **col** converts white space to tabs wherever doing so might shorten printing time.

Related Information

The following commands: “**nroff**, **troff**” on page 709 and “**tbl**” on page 1053.

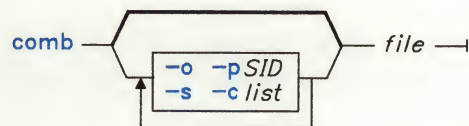
The discussion of **col** in *Text Formatting Guide*.

comb

Purpose

Combines SCCS deltas.

Syntax



OL805098

Description

The **comb** command writes to standard output a shell procedure that can combine the specified deltas (*SIDs*) or all deltas into one delta. You may reduce the size of your SCCS file by running the resulting procedure on the file. You can see how much the file will be reduced by running **comb** with the **-s** flag. If you specify a directory in place of *file*, **comb** performs the requested actions on all SCCS files (that is, those with file names with the **s.** prefix). If you specify a - (minus) in place of *file*, **comb** reads standard input and interprets each line as the name of an SCCS file. **comb** continues to take input until it reads END OF FILE (**Ctrl-D**).

If you do not specify any flags, **comb** preserves only leaf deltas and the minimal number of ancestors needed to preserve the tree (see “**delta**” on page 310).

Note: The **comb** command may rearrange the shape of the tree deltas. It may not save any space; in fact, it is possible for the reconstructed file to actually be larger than the original.

Flags

Each flag or group of flags applies independently to each named file.

-clist Specifies a list of deltas (*SIDs*) that the shell procedure will preserve (see **get -i list** for the *SID* list format on page 482). The procedure will combine all other deltas.

comb

- o Accesses the reconstructed file at the release of the delta to be created for each **get -e** generated; otherwise accesses the reconstructed file at the most recent ancestor. Using the **-o** flag may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file.
- pSID Specifies the *SID* of the oldest delta for the resulting procedure to preserve. All older deltas are combined in the reconstructed file.
- s Causes **comb** to generate a shell procedure that produces a report for each file giving: the file name, size (in blocks) after combining, original size (also in blocks), and percentage change computed by the formula:
$$100 * (\text{original} - \text{combined}) / \text{original}$$

You should run **comb** using this flag and run its procedure before combining SCCS files in order to judge how much space will actually be saved by the combining process.

Files

- | | |
|--------|--|
| s.COMB | The name of the reconstructed SCCS file. |
| comb* | Temporary files. |

Related Information

The following commands: “**admin**” on page 41, “**delta**” on page 310, “**get**” on page 477, “**help**” on page 513, and “**prs**” on page 781.

The **sccsfile** file in *AIX Operating System Technical Reference*.

The discussion of SCCS in *AIX Operating System Programming Tools and Interfaces*.

comm

Purpose

Selects or rejects lines common to two sorted files.

Syntax

`comm` —

one of		
-1	-2	-3
-12	-13	-23
-123		

 — *file1* — *file2* —

OL805099

Description

The **comm** command reads *file1* and *file2* and writes, by default, a three-column output to standard output. The columns consist of:

- Lines that are only in *file1*
- Lines that are only in *file2*
- Lines that are in both *file1* and *file2*.

If you specify - (minus) for one of the file names, **comm** reads standard input. Both *file1* and *file2* should be sorted according to the collating sequence specified by the environment variable **NLCTAB** (see “**ctab**” on page 257 and “**sort**” on page 958),

Flags

- 1 Suppresses the display of the first column (lines in *file1*).
- 2 Suppresses the display of the second column (lines in *file2*).
- 3 Suppresses the display of the third column (lines common to *file1* and *file2*).

Note: Specifying **-123** does nothing (a noop).

Examples

1. To display the lines unique to each file and common to both:

```
comm things.to.do things.done
```

comm

If the files `things.to.do` and `things.done` contain:

<code>things.to.do</code>	<code>things.done</code>
buy soap	2nd revision
groceries	interview
luncheon	luncheon
meeting at 3	system update
system update	tech. review
tech. review	weekly report

then **comm** displays:

```
      2nd revision
buy soap
groceries
      interview
      luncheon
meeting at 3
      system update
      tech. review
      weekly report
```

The first column contains the lines found only in `things.to.do`. The second column, indented with a tab character, lists the lines found only in `things.done`. The third column, indented with two tabs, lists the lines common to both.

2. To display the lines that appear in only one file:

```
comm -23 things.to.do things.done
```

This suppresses the second and third columns of the **comm** listing. If the files are the same as in Example 1, then the following is displayed:

```
buy soap
groceries
meeting at 3
```

Related Information

The following commands: “**cmp**” on page 177, “**ctab**” on page 257, “**diff**” on page 320, “**sdiff**” on page 883, “**sort**” on page 958, and “**uniq**” on page 1118.

The **environment** miscellaneous facility in *AIX Operating System Technical Reference*.

“Overview of International Character Support” in *Managing the AIX Operating System*.

Description

The **comp** command is used to create and modify messages. **comp** is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

By default, **comp** copies a message form to a new draft message and invokes an editor. You can then fill in the message header fields **To:** and **Subject:**, fill in or delete the other header fields (such as **cc:** and **Bcc:**), and add the body of the message. When you exit the editor, the **comp** command invokes the MH command **whatnow**. You can specify any of the **whatnow** subcommands, or you can press **Enter** to see a list of the subcommands. These subcommands enable you to continue composing the message, direct the disposition of the message, or end the processing of the **comp** command. See “**whatnow**” on page 1215 for a description of the subcommands.

You can specify the *form*, or format, of the message by using the **-form** flag or the **+folder** flag. If you do not specify one of these flags, **comp** uses your default message format located in the file *user-mh-directory/components*. If this file does not exist, **comp** uses the system default message format located in */usr/lib/mh/components*.

You can compose a new message, or you can specify the **-use** flag and continue composing an existing message. The **-file**, **-draftfolder**, and **-draftmessage** flags enable you to specify the new or existing message that you want to compose.

Note: The line of dashes or a blank line must be left between the header and the body of the message for the message to be identified when it is sent.

Flags

-draftfolder +folder Places the draft message in the specified folder. If you do not specify this flag, **comp** selects a default draft folder according to the information supplied in the MH profiles. You can define a default draft folder in *\$HOME/.mh-profile*. If **-draftfolder +folder** is followed by *msg*, *msg* represents the **-draftmessage** attribute.

-draftmessage msg Specifies the draft message. You can specify one of the following message references as *msg*:

<i>num</i>	<i>sequence</i>	first
prev	cur	.
next	last	new

If the **-use** flag is specified, the default draft message is **cur**. Otherwise, the default draft message is **new**.

- editor *cmd*** Specifies that *cmd* is the initial editor for composing the message. If you do not specify this flag, **comp** selects a default editor or suppresses the initial edit, according to the information supplied in the MH profiles. You can define a default initial editor in **\$HOME/.mh-profile**.
- file *file*** Places the draft message in the specified file. If you do not specify the absolute path name for *file*, **comp** places *file* in *user-mh-directory*. If *file* exists, **comp** prompts you for the disposition of the draft.
- +folder *msg*** Uses the form of the specified message in the specified folder. You can specify one of the following message references as *msg*:
- | | | |
|-------------|-----------------|--------------|
| <i>num</i> | <i>sequence</i> | first |
| prev | cur | |
| next | last | |
- The default message is the current message in the current folder.
- form *file*** Uses the form contained in the specified file. **comp** treats each line in *file* as a format string.
- help** Displays help information for the command.
- nodraftfolder** Places the draft in the file *user-mh-directory/draft*.
- noedit** Suppresses the initial edit.
- nouse** Creates a new message.
- nowhatnowproc** Does not invoke a program that guides you through the composing tasks. The **-nowhatnowproc** flag also prevents any edit from occurring.
- use** Continues composing an existing draft of a message.
- whatnowproc *cmdstring*** Invokes *cmdstring* as the program to guide you through the composing tasks. See “**whatnow**” on page 1215 for information about the default **whatnow** program and its subcommands.
- Note:** If you specify *whatnow* for *cmdstring*, **comp** invokes an internal **whatnow** procedure rather than a program with the file name **whatnow**.

Profile Entries

Draft-Folder:	Sets your default folder for drafts.
Editor:	Sets your default initial editor.
fileproc:	Specifies the program used to refile messages.
Msg-Protect:	Sets the protection level for your new message files.
Path:	Specifies your <i>user-mh-directory</i> .
whatnowproc:	Specifies the program used to prompt What now? questions.

Files

<i>/usr/lib/mh/components</i>	The system default message form.
<i>user-mh-directory/components</i>	The user's default message form. (If it exists, it overrides the system default message form.)
<i>\$HOME/.mh-profile</i>	The MH user profile.
<i>user-mh-directory/draft</i>	The draft file.

Related Information

Other MH commands: “**ali**” on page 48, “**dist**” on page 336, “**forw**” on page 438, “**prompter**” on page 778, “**repl**” on page 821, “**refile**” on page 817, “**send**” on page 893, “**whatnow**” on page 1215, “**whom**” on page 1222.

The **mh-alias**, **mh-format**, and **mh-profile** files in *AIX Operating System Technical Reference*.

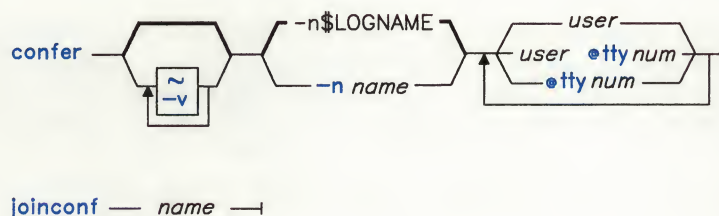
The “Overview of the Message Handling Package” in *Managing the AIX Operating System*.

confer

Purpose

Provides an online conferencing system.

Syntax



OL805174

Description

The **confer** command sets up an online, written conference among logged-in users on your local node. You start a conference by running the **confer** command, specifying the *users* and/or work stations (*@tty num*) that are part of the conference. If the users you specify are logged in and their work stations are writable, they are requested to join the conference by using the **joinconf** command. The other conferees are informed as each user joins the conference.

Once you join a conference, everything you enter at your work station displays at all other work stations that are part of the conference. This display continues until you press **Ctrl-D** to end your own active participation or until you **excuse** a conference participant, thus stopping the display of your contributions at his work station. (See page 190.)

To prevent the confusion that can be caused by several conferees typing at the same time, users should follow some agreed on protocol. The following is one recommended protocol:

- In order to take the floor, a user presses the **Enter** key before entering his contribution. This notifies other participants that he has the floor because his name displays in brackets at their respective work stations.
- A user is presumed to have the floor until he relinquishes it by entering a blank line.
- If two or more users try to claim the floor at the same time, the last person to do so (the one whose name appears last), is assumed to have the floor. The others should immediately relinquish the floor by typing single blank lines.

confer

The **confer** command gives each conference a unique name, normally the name of the conference leader, with additional letters added to it, if necessary. The conference leader can override this default by specifying the **-n** flag.

A user who is logged in to more than one work station is normally written to on all of them, unless the conference leader specifies one of the work stations with the **@ttynum** flag when he invokes **confer**.

A conferee ends his active participation by pressing **Ctrl-D**. This action causes his name and the word **BYE** to display at the work stations of the other conference participants. However, the contributions of the other participants will continue to display at his work station until the other participants each **excuse** him.

You can run shell commands from within a conference by simply prefixing them with a **|** (vertical bar) or an **!** (exclamation point). Using the exclamation point causes the command to run in the normal fashion; the output displays only at the work station that runs it. Using the vertical bar, however, causes the command and all of its standard output and standard error output to become part of the conference, visible to all conferees.

Three subcommands are run directly by **confer** and **joinconf**. These are:

- !excuse name . . .** Excuses the specified conferees from the conference. No further conference material displays at these work stations.
- !~** Makes all contributions from the user who issues it off the record until he issues the **!~~** subcommand.
- !~~** Cancels a preceding **!~**, placing the user's remarks back on the record.

Unless the conference leader makes a conference off the record by specifying the **~** flag, **confer** makes a transcript of all conference proceedings. When a participant leaves the conference, he is asked whether he wants a transcript. If he does, he is mailed a copy when the conference concludes. Any participant can make a comment off the record in a conference that is otherwise **on the record** by beginning the line with a **~** (tilde).

Conference contributions are normally transmitted one line at a time. If the conference leader specifies the **-v** flag, transmission occurs one character at a time. As this mode of transmission sends all user typing errors and hesitations and imposes a considerably larger load on the system, its use is strongly discouraged.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

- nname** Assigns *name* to the conference transcript. The conference name is used by those joining the conference so that they get into the right one. The name of the user who starts the conference is the default conference name.
- v** Transmits conference messages one character at a time.
- ~** Sets up the conference *off the record*, that is, no transcript of the proceedings is recorded.
- @ttynum** Specifies a particular work station for a conferee, if a *user* is also specified (for example, `tty1`). This is useful if a conferee is logged in to more than one work station. If no *user* is specified, this flag invites any user logged in to the specified work station to participate.

Examples

1. To start a conference with `steve` and `rachel`:

```
confer steve rachel
```

Running the **confer** command makes you the conference leader, so your login name is also the name of the conference. **confer** sends `steve` and `rachel` a message inviting them to join your conference and giving them the conference name.

2. To specify work stations that may join the conference:

```
confer steve@tty5 rachel @tty10
```

Suppose that `steve` is logged in at the work stations `tty3`, `tty4`, and `tty5`, and that `rachel` is logged in at `tty7` and `tty8`. This command invites `steve` to join the conference at work station `tty5` only, invites `rachel` to join at either work station she is using or at both, and invites whoever is logged in at `tty10` to join.

3. To join a conference named `paula`:

```
joinconf paula
```

Now the text you type becomes part of the dialog: prefixed with your name, displayed at each participant's work station, and recorded in the transcript of the conference.

4. Suppose that you start a conference by entering the command given in Example 2, and the person using `tty10` decides not to join the conference. If you do nothing, this person also sees the dialog, even though not participating in it. To prevent this from happening, each person that has joined the conference must enter:

```
!excuse @tty10
```


confer

Similarly, if `rachel` decides to join the conference from `tty7`, the discussion is also displayed at her other work station, `tty8`, unless everyone enters:

```
!excuse rachel@tty8
```

rachel should enter this, too, but only at `tty7`, the work station she is using for the conference.

5. To make a single-line statement off the record:

```
~Coffee and donuts at my place.
```

confer displays lines beginning with `~` (tilde) at participants' work stations, but does not include them in the record of the conference.

To make a multiple-line statement:

```
!~  
Everyone is invited  
to my place after the conference  
for coffee and donuts.  
!~~
```

6. To run a shell command privately, without leaving the conference:

```
!li
```

This lists the current directory without including the `li` command or its output in the conference.

7. To include the output of a shell command in the discussion:

```
!cat notes.conf
```

This lists the contents of the file `notes.conf` at each participant's work station, and includes it in the conference record.

8. To send command output to others, off the record:

```
!~  
!cat notes.conf  
!~~
```

9. To leave the conference, press **Ctrl-D**. If your user name is `paula`, then after you press **Ctrl-D**, the message: `[paula] BYE` is sent to the other participants. The rest of the discussion continues to appear at your work station until each of the other participants enters:

```
!excuse paula
```

Files

/etc/utmp	List of logged-in users.
/dev/tty??	Work station names.
/tmp/*.cnf	User transcript files.
/tmp/*.ln?	Links to main conference file.
/tmp/*.mls	Transcript mailing list.

Related Information

The following command: **“write”** on page 1225.

config

config

Purpose

Extracts configuration information from configuration files.

Syntax

```
config -m /etc/master -c conf.c -l specials systemfile  
      -m mfile -c cfile -l spfile
```

OL805416

Description

The **config** program reads the AIX master and system configuration files (by default **/etc/master** and the specified *systemfile*). It writes a C Language configuration file and a special file list (by default **conf.c** and **specials**). The special file list is a list of the **mknod**, **chown**, and **chmod** commands that the shell runs to define the necessary special files. The return code is the number of errors encountered.

The C Language configuration file can then be compiled and linked with other kernel object files to produce a new kernel. Normally, when you want to reconfigure the kernel, you should run the **make** command with the **Makefile** supplied in the **/usr/sys** directory. This runs **config** and then builds a new kernel. For a discussion of reconfiguring the kernel, see *Managing the AIX Operating System*.

Flags

- c cfile** Writes the C configuration file to *cfile* instead of to **conf.c**.
- l spfile** Writes the special file list commands to *spfile* instead of to **specials**.
- m mfile** Reads *mfile* instead of **/etc/master**.

Files

/etc/master	Default master configuration file.
/etc/system	A system configuration file.
conf.c	Default C configuration file.
specials	Default special file list.

Related Information

The following commands: “**make**” on page 625 and “**vrconfig**” on page 1206.

The **master** and **system** files in *AIX Operating System Technical Reference*.

The discussion of **config** in *Managing the AIX Operating System*.

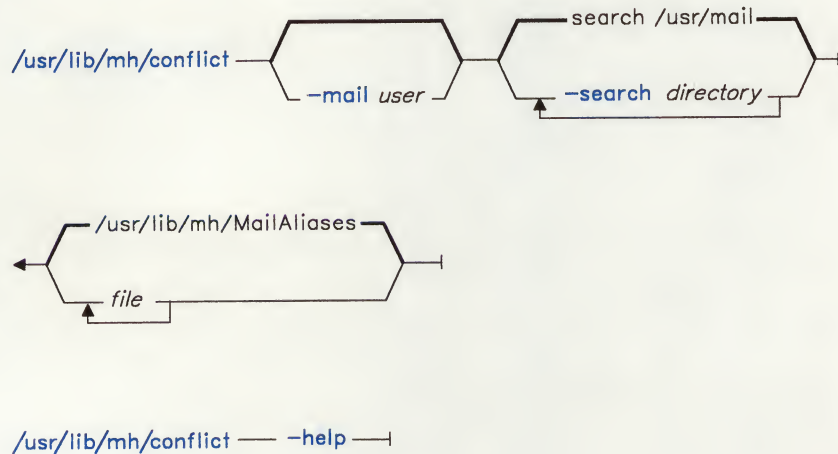
conflict

conflict

Purpose

Searches for alias and password conflicts.

Syntax



AJ2FL225

Description

The **conflict** command is used to find conflicts in aliases and to find invalid mail drops. **conflict** is not designed to be run directly by the user; it is designed to be called by **cron** and other programs used for system accounting. **conflict** is a system administrator command that is usually invoked by its full path name. The **conflict** command is part of the MH (Message Handling) package.

The **conflict** command searches all specified alias files for duplicate alias names that do not resolve to the same address. By default, **conflict** searches **/usr/lib/mh/MailAliases**. **conflict** also searches all specified mail drop directories for mailbox files with names that do not correspond to valid users defined in **/etc/passwd**.

The **conflict** lists its output on the display, unless you specify the **-mail** flag. **-mail** causes **conflict** to mail its output to the specified user.

Flags

- | | |
|--------------------------|---|
| -help | Displays help information for the command. |
| -mail user | Sends the results of the conflict command to the specified user. |
| -search directory | Searches the indicated directories for invalid mailboxes. You can specify any number of -search flags. The default mailbox directory is /usr/mail . |

Files

- | | |
|------------------------------|--------------------------------|
| /usr/lib/mh/mtstailor | The MH tailor file. |
| /etc/passwd | List of users. |
| /etc/group | List of groups. |
| /usr/mail/\$USER | The location of the mail drop. |

Related Information

Other MH commands: “**ali**” on page 48, “**whom**” on page 1222.

The **mh-alias**, **mh-mail**, and **mh-profile** files in *AIX Operating System Technical Reference*.

The “Overview of the Message Handling Package” in *Managing the AIX Operating System*.

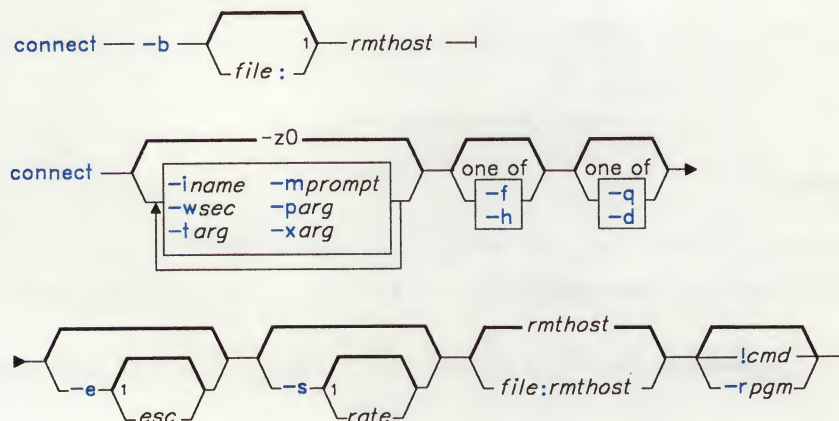
connect

connect

Purpose

Establishes a connection to a remote system.

Syntax



OL805388

Description

The **connect** command lets you establish a connection to a remote host. **connect** runs in two parts. The first part makes the connection with the remote system specified by *rmthost*. The second part is a program called the **talker**. It runs automatically and exchanges data with the *rmthost*. For more information about the **talker** program, see “**connect**” in *AIX Operating System Technical Reference*. Any flags that you specify are passed directly to the **talker** without interpretation. The default **talker** for asynchronous links is **atalk**.

The **connect** command uses a system-wide control file, **connect.con**, located in */usr/lib/INnet*. You can specify an additional control file, *file:rmthost*. If you do not specify an additional file, **connect** searches *\$HOME/bin* for a **connect.con** file. Information needed to complete the connection is found in one of these files.

Attributes needed to complete the connection are taken from the control file or from the command line assignment *var=val*. For a description of the parameters, see “**connect**” in *AIX Operating System Technical Reference*.

When **atalk** detects an escape sequence in the input, it places the work station in its former mode of operation and prompts you with the local prompt. You can then use the flags that follow. Once the flag has run, **atalk** returns to its former mode.

The **connect** command does not limit access to the phone system to control dialing based on the number to be called.

Warning: The **connect** command lets you set up and maintain connections through a wide variety of communications devices. It interacts with you through the file **connect.con** which is free-format. Problems with the format of this file may cause unpredictable results.

Flags

Note: There are no spaces between the flags and the associated parameters.

-b Sends a break to the port. This is done by lowering the transmission speed to 75 bps and transmitting an ASCII NULL on the port. If the speed is too low, less than 100 bps, this may not work.

-d

-q Closes, quits (**q**) or disconnects (**d**) the port. Note that this does *not* end your job or session at the remote site. After closing the port, **connect** exits.

-e[esc] Sets the escape sequence to the character string *esc*. If you do not specify *esc*, **connect** displays escape sequence. It takes the default escape sequence from the environment variable **CONESC**, if defined, or else sets it to:

Ctrl-VuCtrl-M

-f

-h Enables (**-h**) or disables (**-f**) local echoing.

-iname Writes file *name* to the port.

Warning: If you are connected to the remote host by RS-232 lines, data from the file may be lost if the remote host cannot keep up with the input.

Normally, this flag is used to transfer a small file from the local site to the remote site. File transmission must be ended manually by pressing **Ctrl-D**.

connect

For example:

```
cat > newfile  
[escape sequence]  
LOCAL: ifred
```

```
.
```

Ctrl-D

- mprompt** Set the local prompt to the *prompt* character string. **connect** displays this prompt when it recognizes the escape sequence. By default, it sets the prompt to the value of the environment variable **CONPMT**. If this variable is not set, it uses the string **LOCAL:**.
- parg** Sets parity as specified by *arg*, where *arg* is one of the following characters: **o** (odd), **e** (even), **7** (both even and odd), or **8** (eight data bits).
- rpgm** Runs the network program *pgm*. Anything following *pgm* on the command line is passed to *pgm* as an argument, along with the additional arguments **-i3 -o3**. The port set up as file descriptor 3. The program is run as a child process.
- srate** Sets the transmission speed to *rate*, which is one of the following: 0, 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, exta, extb (0 effectively turns off the port). If you do not specify *rate*, current transmission speed displays.
- targ** Enables or disables transcripts. If *arg* is any character string other than a minus or plus sign, the transcript function is enabled with the specified file *arg* as transcript. When you use an existing file as a transcript file, new data is added to its end. Use **t-** to disable the transcript function, and **t+** to enable the transcript to the previous transcript file (no default).
- wsec** Sets the inter-line delay of the include function to cause a delay interval of the specified seconds between each line written to the port. The default value is 0.
- xarg** Enables or disables input or output flow control. If the input flow control is enabled, **CTRL-S** and **CTRL-Q** are automatically sent to the remote host to control the rate at which it transmits data. If the output flow control is enabled, **CTRL-S** and **CTRL-Q** are automatically honored if received from the host. This is useful when using the **include** command. **xi+** enables input flow control. **xi-** disables input flow control. **xi** displays the current state. For control of output flow control, replace **xi** with **xo**. See the discussion of **IXON** and **IXOFF** in the **termio** file in *AIX Operating System Technical Reference*.
- !cmd** Runs the AIX command *cmd*. Anything that follows **!** (exclamation point), including arguments to *cmd*, is passed to the local shell to be run by the **system** system call. In particular, all I/O redirection and piping works.

Files

/usr/lib/INnet/connect.con	System-wide connection control file.
\$HOME/bin/connect.con	Private connection control file.
/usr/lib/INnet/dialers/*	System-wide dialer programs.
\$HOME/bin/*	Private dialer programs.
/usr/lib/INnet/atalk	Default talker program, asynchronous lines.
/etc/sites	Network sites file.
/etc/locks	Directory for locks on ports (devices) used for logins and out-going connections.

Related Information

The **system** and **exec** system calls, the **connect** subroutine, and the **termio** special facility in *AIX Operating System Technical Reference*.

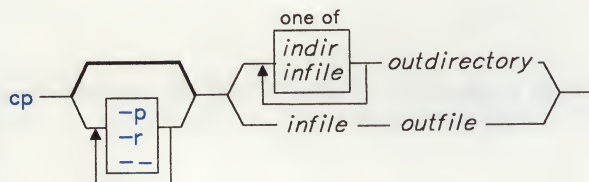
cp

cp

Purpose

Copies files.

Syntax



OL805100

Description

The **cp** (copy) command copies a source file or the files in a source directory to a target file or directory. If your output is to a directory, then the files are copied to that directory with the same file name. If either *infile* or *outfile* is a symbolic link, the link is followed when **cp** is performed. An error message is displayed if the link cannot be followed.

You can also copy special device files. If the file is a named pipe, the data in the pipe is copied into a regular file. If the file is a device, the file is read until the end of file and that data is copied into a regular file.

Notes:

1. Do not name *outfile* as one of the input files.
2. If you specify a directory for the *outfile*, the directory must already exist.
3. If the *infile* contains subdirectories and the subdirectories do not exist, the system creates them.

Flags

- p Preserves the modification times and modes of the *infile* for the copy.
- r Copies each subtree rooted at the *infile* (recursive copy). If the *infile* is a directory, then the *outfile* must be a directory.
- Indicates that the arguments following this flag are to be interpreted as file names. This null flag allows the specification of file names that start with a minus.

Examples

1. To make another copy of a file in the current directory:

```
cp prog.c prog.bak
```

This copies `prog.c` to `prog.bak`. If the file `prog.bak` does not already exist, then `cp` creates it. If it does exist, then `cp` replaces it with a copy of `prog.c`.

2. To copy a file to another directory:

```
cp jones /u/nick/clients
```

This copies `jones` to `/u/nick/clients/jones`.

3. To copy a file to a new file and preserve the modification date and time:

```
cp -p smith smith.jr
```

This copies `smith` to `smith.jr`. Instead of creating the file with the current date and time stamp, the system gives `smith.jr` the same date and time as `smith`.

4. To copy all the files in a directory to a new directory:

```
cp /u/nick/clients/* /u/nick/customers
```

This copies the files and directories in the directory `clients` to the directory `customers`.

5. To copy a directory, its files and its subdirectories to another directory:

```
cp -r /u/nick/clients /u/nick/customers
```

This copies the directory `clients`, its files, its subdirectories, and the files in the subdirectories to the directory `customers`.

6. To copy a specific set of files to another directory:

```
cp jones lewis smith /u/nick/clients
```

This copies `jones`, `lewis`, and `smith` to `/u/nick/clients`.

7. To use pattern-matching characters to copy files:

```
cp programs/*.c .
```

This copies the files in directory `programs` that end with `.c` to the current directory (`.`). You must type a space between the `c` and the final period.

Related Information

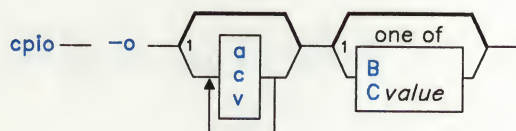
The following commands: “**cpio**” on page 205, “**link, unlink**” on page 575, “**ln**” on page 581, and “**mv**” on page 679.

cpio

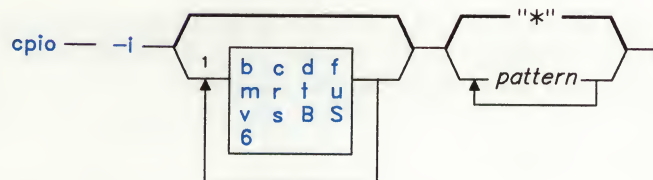
Purpose

Copies files into and out of archive storage and directories.

Syntax



OL805175



OL805350



¹ Do not put a blank between these items.

OL805351

Description

Warning: If you redirect the output from **cpio** to a special file (device), you should redirect it to the raw device and not the block device. Because writing to a block device is done asynchronously, there is no way to know if the end of the device has been reached.

cpio -o

This command reads file path names from standard input and copies these files to standard output along with path names and status information. Path names cannot exceed 128 characters. Avoid giving **cpio** path names made up of many unique linked files as it may not have enough memory to keep track of them and so would lose linking information.

cpio -i

This command reads from standard input an archive file created by the **cpio -o** command and copies from it the files with names that match *pattern*. These files are copied into the current directory tree. You may list more than one *pattern*, using the file name notation described under “**sh**” on page 913. Note, however, that in this application the special characters * (asterisk), ? (question mark), and [. . .] (ellipse) match the / (slash) in path names, in addition to their use as described under “**sh**” on page 913. The default *pattern* is * (select all files in the current directory). In an expression such as [a-z], the minus means “through” according to the current collating sequence.

A collating sequence may define *equivalence classes* for use in character ranges. See the “Overview of International Character Support” in *Managing the AIX Operating System* for more information on collating sequences and equivalence classes.

Japanese Language Support Information

Note: A collating sequence in Japanese Language Support does not define equivalence classes for use in range expressions. To avoid unpredictable results when using a range expression to match a class of characters, use a *character class expression* rather than a standard range expression. For information about character class expressions, see the discussion of this topic in “**ed**” on page 371.

cpio -p

This command reads file path names from standard input and copies these files into the named *directory*. The specified *directory* must already exist. If these path names include directory names and if these directories do not already exist, you must use the **d** flag to cause the *directory* to be created.

Note: You can copy special files only if you have superuser authority.

Flags

All flags must be listed together, without any blanks between them. Not all of the following flags can be used with each of the **-o**, **-i**, and **-p** flags.

- a** Resets the access times of copied files to the current time.
- b** Swaps both bytes and halfwords.
 Note: If there are an odd number of bytes or halfwords in the file being processed, data can be lost.
- B** Performs block input/output, 5120 bytes to a record.
- c** Writes header information in ASCII character form.
- Cvalue** Performs block input/output, *value* * 512 bytes to a record.
 Note: The **C** flag and the **B** flag are mutually exclusive. If you list both, **cpio** uses the last one it encounters in the flag list.
- d** Creates directories as needed.
- f** Copies all files except those matching *pattern*.
- l** Links files rather than copies them, whenever possible. This flag is usable only with **cpio -p**.
- m** Retains previous file modification time. This flag does not work when copying directories.
- r** Renames files interactively. If you do not want to change the file name, enter the current file name or press the **Enter** key only. In this last case, **cpio** does not copy the file.
- s** Swaps bytes. This flag is usable only with **cpio -i**.
 Note: If there are an odd number of bytes in the file being processed, data can be lost.
- S** Swaps halfwords. This flag is usable only with **cpio -i**.
 Note: If there are an odd number of halfwords in the file being processed, data can be lost.
- t** Creates a table of contents. This does not copy any files.
- u** Copies unconditionally. An older file now replaces a newer file with the same name.
- v** Lists file names. If you use this with the **t** flag, the output looks similar to that of the **ls -l** command.
- 6** Processes an old file (one written in UNIX Sixth Edition format). This flag is usable only with **cpio -i**.

Examples

1. To copy files onto diskette:

```
cpio -ov <filenames >/dev/rfd0
```

This copies the files with path names that are listed in the file `filenames` in a compact form onto the diskette (`>/dev/rfd0`). The `-v` flag causes **cpio** to display the name of each file as it is copied. This command is useful for making backup copies of files. The diskette must already be formatted, but it must not contain a file system or be mounted.

2. To copy files in the current directory onto diskette:

```
ls *.c | cpio -ov >/dev/rfd0
```

This copies all the files in the current directory whose names end with `.c`.

3. To copy the current directory and all subdirectories onto diskette:

```
find . -print | cpio -ov >/dev/rfd0
```

This saves the directory tree that starts with the current directory (`.`) and includes all of its subdirectories and files. A faster way to do this is:

```
find . -cpio /dev/rfd0 -print
```

The `-print` displays the name of each file as it is copied.

4. To list the files that have been saved onto a diskette with **cpio**:

```
cpio -itv </dev/rfd0
```

This displays the table of contents of the data previously saved onto `/dev/rfd0` in **cpio** format. The listing is similar to the long directory listing produced by `li -l`. To list only the file path names, use only the `-it` flags.

5. To copy the files previously saved with **cpio** from a diskette:

```
cpio -idmv </dev/rfd0
```

This copies the files previously saved onto `/dev/rfd0` by **cpio** back into `(-i)` the file system. The `-d` flag allows **cpio** to create the appropriate directories if a directory tree was saved. The `-m` flag maintains the last modification time that was in effect when the files were saved. The `-v` causes **cpio** to display the name of each file as it is copied.

6. To copy selected files from diskette:

```
cpio -i "*.c" "*.o" </dev/rfd0
```

This copies the files that end with .c or .o from diskette. Note that the patterns "*.c" and "*.o" must be enclosed in quotation marks to prevent the shell from treating the * as a pattern-matching character. This is a special case in which **cpio** itself decodes the pattern-matching characters.

7. To rename files as they are copied from diskette:

```
cpio -ir </dev/rfd0
```

The **-r** flag causes **cpio** to ask you whether or not to rename each file before copying it from diskette. For example, the message:

```
Rename <prog.c>
```

asks whether to give the file saved as prog.c a new name as it is copied in. To rename the file, type the new name and press **Enter**. To keep the same name, you must enter the name again. To avoid copying the file at all, press the **Enter** key alone.

8. To copy a directory and all of its subdirectories:

```
mkdir /u/jim/newdir
find . -print | cpio -pdl /u/jim/newdir
```

This duplicates the current directory tree, including the current directory and all of its subdirectories and files. The duplicate is placed in the new directory /u/jim/newdir. The **-l** flag causes **cpio** to link files instead of copying them, when possible.

Related Information

The following commands: “**ar**” on page 55, “**find**” on page 422, and “**ln**” on page 581.

The **cpio** system call in *AIX Operating System Technical Reference*.

“Overview of International Character Support” in *Managing the AIX Operating System*.

The discussion of Japanese Language Support in *Japanese Language Support User's Guide*.

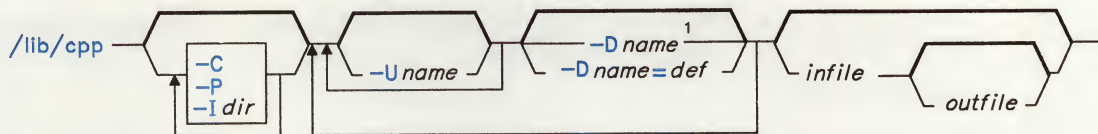
cpp

cpp

Purpose

Performs file inclusion and macro substitution on C Language source files.

Syntax



¹ The default *def* is 1.

OL805378

Description

The **cpp** program is the C Language preprocessor. It reads *infile* and writes to *outfile* (standard input and standard output by default). Although you can use this preprocessor by itself, it is best to use it through the **cc** command, which by default sends a C Language source file to **cpp** as the first pass in compilation.

The **cpp** program recognizes two special names, **--LINE--** (the current line number) and **--FILE--** (current file name). These names can be used anywhere just as any other defined name.

All **cpp** directive lines must begin with a hash sign (#). These directives are:

#define *name token-string*

Replaces subsequent instances of *name* with *token-string*.

#define *name(arg, . . . ,arg) token-string*

Replaces subsequent instances of the sequence *name (arg, . . . ,arg)* with *token-string*, where each occurrence of an *arg* in *token-string* is replaced by the corresponding token in the comma-separated list. Note that there must not be any space between *name* and the left parenthesis.

#undef *name*

Ignores the definition of *name* from this point on.

- #include "file"**
#include <file> Includes at this point the contents of *file*, which **cpp** then processes.
- If you enclose *file* in " ", (double quotation marks) **cpp** searches first in the directory of *infile*, second in directories named with the **-I** flag, and last in directories on a standard list .
- If you use the <*file*> notation, **cpp** searches for *file* only in the standard places. It does not search the directory in which *infile* resides.
- #line num ["file"]** Includes line control information for the next pass of the C compiler. *num* is the line number of the next line and *file* is the file from which it comes. If you omit "*file*", the current file name remains unchanged.
- #endif** Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**.
- #ifdef name** Places the subsequent lines in the output only if *name* has been defined by a previous **#define** and has not been undefined by an intervening **#undef**.
- #ifndef name** Places the subsequent lines in the output only if *name* has not been defined by a previous **#define** or has been undefined by an intervening **#undef**.
- #if expr** Places subsequent lines in the output only if *expr* evaluates to nonzero. All the binary nonassignment C operators, the **?:** operator, and the unary **-**, **!**, and **~** operators are legal in *expr*. The precedence of the operators is the same as that defined in the C Language. There is also a unary operator **defined**, which can be used in *expr* in these two forms:
- defined (name)**
defined name
- This allows the utility of **#ifdef** and **#ifndef** in a **#if** directive. Only these operators, integer constants, and names which are known by **cpp** should be used in *expr*. The **sizeof** operator is not available.
- #else** Places subsequent lines in the output only if the expression in the preceding **#if** directive evaluates to False (and hence the lines following the **#if** and preceding the **#else** have been ignored).
- You can nest the test directives and the possible **#else** directives.

Flags

-C	Copies source file comments to the output file. If you omit this flag, cpp removes all comments (except those found on cpp directive lines).
-Dname[=def]	Defines <i>name</i> as in a #define directive. The default <i>def</i> is 1.
-Idir	Looks first in <i>dir</i> , then looks in the directories on the standard list for #include files with names that do not begin with a / (slash). See the previous discussion of #include .
-P	Preprocesses input without producing line control information for the next pass of the C compiler.
-Uname	Removes any initial definition of <i>name</i> , where <i>name</i> is a reserved symbol predefined by the preprocessor.

Examples

1. To display the text that the preprocessor sends to the C compiler:

```
/lib/cpp pgm.c
```

This preprocesses `pgm.c` and displays the resulting text at the work station. You may want to see the preprocessor output when looking for errors in your macro definitions.

2. To create a file containing more readable preprocessed text:

```
/lib/cpp -P -C pgm.c pgm.i
```

This preprocesses `pgm.c` and stores the result in `pgm.i`. It omits line numbering information intended for the C compiler (**-P**), and includes program comments (**-C**).

3. To predefine macro identifiers:

```
/lib/cpp -DBUFFERSIZE=512 -DDEBUG pgm.c pgm.i
```

This defines `BUFFERSIZE` with the value 512 and `DEBUG` with the value 1 before preprocessing.

4. To use **#include** files located in nonstandard directories:

```
/lib/cpp -I/u/jim/include pgm.c
```

This looks in the current directory for quoted **#include** files, then in `/u/jim/include`, and then in the standard directories. It looks in `/u/jim/include` for angle-bracketed **#include** files (`< >`) and then in the standard directories.

Files

/usr/include Standard directory for **#include** files.

Related Information

The following commands: “**cc**” on page 140 and “**m4**” on page 603.

craps

craps

Purpose

Plays craps.

Syntax

`/usr/games/craps` —

OL805188

Description

The **craps** game plays a form of the game of craps that is played in Las Vegas. It simulates the **roller** while you place bets. Bet with the roller by making a positive bet or with the **House** by making a negative bet.

You start with a \$2000 bankroll. When the program prompts with `bet?`, you may bet all or part of your bankroll. If you bet more than your bankroll, the program repeats the prompt until you make a legal bet. Then the roller throws the dice. The payoff odds are one to one. The player wins depending on whether the bet is placed with the roller or with the House. The first roll is the roll immediately following a bet.

The following rules apply. On the first roll, 7 or 11 wins for the roller; 2, 3, or 12 wins for the House; and any other number becomes the **point** and you roll again (the next rule then applies). On subsequent rolls, the point wins for the roller; 7 wins for the House; and any other number rolls again.

If you lose your bankroll, the House prompts `marker?`, offering to lend you an additional \$2000. Accept the loan by responding `y` or `yes`. Any other response ends the game. When you hold markers, the House reminds you before a bet how many markers are outstanding. When you have markers and your bankroll exceeds \$2000, **craps** asks `Repay marker?` If you want to repay part or all of your loan, respond with `y` (or `yes`). If you have more than one marker, **craps** asks you `How many?` If you respond with a number greater than the number of markers you hold, it repeats the prompt until you enter a valid number. If you accumulate 10 markers (a total loan of \$20,000), **craps** tells you so and exits. If you accumulate a bankroll of more than \$50,000 while holding markers, the money owed is repaid automatically.

A bankroll of more than \$100,000 breaks the bank, and **craps** will prompt `New game?` To quit the game, press INTERRUPT (**Alt-Pause**); **craps** displays whether you have won, lost, or broken even and exits.

crash

Purpose

Examines system images.

Syntax

```
crash - /dev/mem
      - system
```

OL805101

Description

The **crash** command is an interactive utility for examining an operating system image (a core image or the running kernel). It has facilities for interpreting and formatting control structures in the system and certain miscellaneous functions useful for examining a dump.

The *system* parameter specifies the file that contains the system image and the kernel symbol definitions. You can run **crash** with no arguments to examine an active system. The default value is **/dev/mem**. If you specify a system-image file, **crash** assumes it is a system dump file and sets the default process to the process running at the time of the crash.

Notes:

1. When using **crash** to identify the flags it uses, a source listing of system header files may be helpful.
2. Stack tracing of the current process on a running system does not work.

The **crash** command recognizes the following aliases in subcommand *format* specifications.

Format	Aliases	Format	Aliases	Format	Aliases
byte	b	hexadecimal	hexadec, hex, h, x	octal	oct, o
character	char, c	inode	ino, i	write	w
decimal	dec, e	longdec	ld, D		
directory	direct, dir, d	longoct	lo, O		

Subcommands

The **crash** command presents a prompt (>) when it is ready to interpret subcommands entered at the work station. The general subcommand format for **crash** is:

subcommand [flags] [structures to be displayed]

When allowed, *flags* modify the format of the data displayed. If you do not specify which structure elements you want to examine, all valid entries are displayed. In general, those subcommands that perform I/O with addresses assume hexadecimal notation.

Most of the subcommands recognized by **crash** have *aliases* (abbreviated forms that give the same result). **crash** recognizes the following subcommands:

buf [*buffer-header*] . . .

Displays the system buffer headers.

buffer [*format*] [*buffer*] . . .

Displays the data in a system buffer according to *format*. If you do not provide a *format* parameter, the previous *format* is used. Valid formats include **decimal**, **octal**, **hex**, **character**, **byte**, **directory**, **i-node** and **write**. The **write** format creates a file in the current directory containing the buffer data.

callout

Aliases: **calls**, **call**, **c**, **timeout**, **time**, **tout**

Displays all entries in the callout table.

cm [*slot-number segment-number*]

If you specify the process slot-number and segment number, this subcommand changes the map of **crash** internal pointers for any segment of a process not swapped out. This allows the **od** subcommand to display data relative to the beginning of the segment desired. If you enter **cm** without any parameters, **cm** resets the map (equivalent of a **reset** subcommand). Use only when analyzing the currently running system.

ds [*data-address*] . . .

Finds the data symbols closest to the given addresses.

du [*slot-number*]

Uses the specified process slot number to display a combined hex and ASCII dump of the user block for any process that has not been swapped out. The default is the current process.

- file** [*file-table-entry*] . . . Aliases: **files**, **f**
 Displays the file table. Unless specific file entries are requested, only those with a nonzero reference are displayed.
- fs** [*slot-number*]
 Traces a kernel stack for the process specified by the process slot number for any process that has not been swapped out. Displays the called subroutines with a hex dump of the stack frame for the subroutine which contains the parameters passed to the subroutine. The default process is the currently running process.
- inode** [-] [*i-node-table-entry*] . . . Aliases: **ino**, **i**
 Displays the i-node table. The - flag also displays the i-node data block addresses. Unless specific i-node entries are requested, only those with a nonzero reference are displayed.
- map** [*map-name*] . . .
 Displays the named system map structures.
- mount** [*mount-table-entry*] . . . Aliases: **mnt**, **m**
 Displays the mount table. Unless specific mount table entries are requested, only those in use are displayed.
- nm** [*symbol*] . . .
 Displays symbol value and type as found in the *kernel-image* file.
- od** [*symbol name or address*] [*count*] [*format*]
 Dumps *count* data values starting at the symbol value or address given according to *format*. Allowable formats are **octal**, **longoct**, **decimal**, **longdec**, **character**, **hex** or **byte**.
- proc** [-] [-r] [*process-table-entry*] . . . Aliases: **ps**, **p**
 Displays the process table. (See the */usr/include/sys/proc.h* file for this structure definition.) The -r flag causes only runnable processes to be displayed. The - (minus) alone displays a longer listing.
- q**
 Exits from **crash**.
- reset** Aliases: **r**
 Reinitializes the **crash** data, takes another slice from */dev/mem*, and updates the process table. Any new processes created can be displayed. Use only when analysing the currently running system.

crash

- stack** [*process-table-entry*] . . . Aliases: **stk, s, kernel, k**
Displays a dump of the kernel stack of a process. The addresses shown are virtual data addresses rather than true physical locations. If you do not specify an entry, information about the last running process is displayed. You can not trace the stack of the current process on a running system.
- stat**
Displays statistics found in the dump. These include the panic message (if a panic occurred), time of crash, and system name.
- text** [*text-table-entry*] . . . Aliases: **txt, x**
Displays the text table. Unless specific text entries are requested, only those with a nonzero i-node pointer are displayed.
- trace** [*process-table-entry*] . . . Aliases: **t**
Displays a kernel stack trace of the current process. The trace starts at the bottom of the stack and attempts to find valid stack frames deeper in the stack. If you do not specify a process table entry, information about the last running process is displayed.
- ts** [*text-address*] . . .
Finds the text symbols closest to the given addresses.
- tty** [*type*] [-] [*tty-entry*] . . . Aliases: **term, dz, dh**
Displays the tty structures. The *type* parameter specifies which structure is used (such as **ksr**, or **rs**). The last *type* entered with the **tty** command becomes the default. The - flag displays the **stty** parameters for the given line.
- user** [*process-table-entry*] . . . Aliases: **uarea, u-area, u**
Displays the user structure of the named process as determined by the information contained in the process table entry. (See the **/usr/include/sys/user.h** file for this structure definition.) If you do not specify the entry, the information about the last running process is displayed. Attempting to display a paged process produces an error message.
- var** Aliases: **tunables, tunable, tune, v**
Displays the tunable system parameters.
- vfs** [-] [*vfs slot-number*]
Uses the specified vfs slot number to display an entry in the vfs table. The - flag displays the vnodes associated with the vfs. The default is to display the entire vfs table.
- vnode** [*vnode slot-number*]
Uses the specified vnode slot number to display an entry in the vnode table. The default is to display the entire vnode structure.

- !
Runs shell commands.
- ?
Displays summary of **crash** commands.

Files

/usr/include/sys/*.h	Header files for table and structure information.
/dev/mem	Default system-image file.
/unix	Default kernel-image file.
buf.#	Files containing buffer data.

Related Information

The following commands: “**mount**” on page 669, “**nm**” on page 705, “**ps**” on page 786, “**sh**” on page 913, and “**stty**” on page 1018.

cron

cron

Purpose

Runs commands automatically.

Syntax

`cron` —¹

¹ Not usually run from the command line, but included in `/etc/rc`.

OL805184

Description

The **cron** command runs shell commands at specified dates and times. Regularly scheduled commands can be specified according to instructions contained in **crontab** files. You can submit your **crontab** file via the **crontab** command (see page 222). Use the **at** command (see page 63) to submit commands that are to be run only once. Because **cron** never exits, it should be run only once. This is best done by running **cron** from the initialization process through the `/etc/rc` command file (see page 806).

The **cron** command examines **crontab** files and **at** command files only during process initialization and when a file changes. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

The **cron** command also executes a **sync** system call approximately once a minute to assure that all information in memory that should be on disk (buffered output) is written out. These periodic updates minimize the possibility of file system damage in the event of a crash. In addition, **cron** keeps a number of frequently used system directories open to keep their i-nodes in kernel memory for faster access.

The **cron** command creates a log of its activities in `/usr/lib/cron/log`.

For a discussion of how to schedule commands, see “**crontab**” on page 222.

Files

<code>/usr/lib/cron</code>	Main cron directory.
<code>/usr/lib/cron/log</code>	Accounting information.
<code>/usr/spool/cron</code>	Spool area.
<code>/bin</code>	Directory kept open.
<code>/lib</code>	Directory kept open.

/usr	Directory kept open.
/usr/bin	Directory kept open.
/usr/lib	Directory kept open.
/etc	Directory kept open.
/tmp	Directory kept open.

Related Information

The following commands: “**at**, **batch**” on page 63, “**crontab**” on page 222, and “**rc**” on page 806.

The **sync** system call and the **crontab** file in *AIX Operating System Technical Reference*.

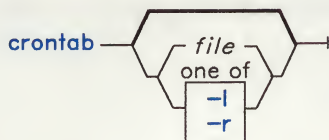
crontab

crontab

Purpose

Submits a schedule of commands to **cron**.

Syntax



OL805003

Description

The **crontab** command copies the specified *file*, or standard input if you do not specify a *file*, into a directory that holds all users' **crontab** files. The **cron** command runs commands according to the instructions in these **crontab** files. It then mails you the output from standard output and standard error for these commands, unless you redirect standard output or standard error. When entries are made to a **crontab** file, all previous entries are erased.

You may use **crontab** if your logname appears in the file **/usr/lib/cron/cron.allow**. If that file does not exist, **crontab** checks the file **/usr/lib/cron/cron.deny** to determine if you should be denied access to **crontab**. If neither file exists, you can submit a job only if you are operating with superuser authority. The allow/deny files contain one user name per line.

Notes:

1. If your login ID is associated with more than one login name, **crontab** uses the first login name that appears in the **/etc/passwd** file, regardless of which login name you might actually be using.
2. If **cron.allow** exists, the superuser's log name must appear there for the superuser to be able to use the command.

Each **crontab** file entry consists of a line with six fields, separated by spaces and tabs, that contain, respectively:

1. The minute (0-59)
2. The hour (0-23)
3. The day of the month (1-31)
4. The month of the year (1-12)
5. The day of the week (0-6 for Sunday-Saturday)
6. The shell command.

Each of these fields can contain:

- A number in the specified range
- Two numbers separated by a minus to indicate an inclusive range
- A list of numbers separated by commas, which selects all numbers in the list
- An asterisk, meaning all legal values.

Note that the specification of days may be made by two fields (day of the month and day of the week). If you specify both as a list of elements, both are adhered to. For example, the following entry:

`0 0 1,15 * 1 command`

would run *command* on the first and fifteenth days of each month, as well as every Monday. To specify days by only one field, the other field should contain an `*`.

The **cron** command runs the command named in the sixth field at the selected date and time. If you include a `%` (percent sign) in the sixth field, **cron** treats everything that precedes it as the command invocation and makes all that follows it available to standard input, unless you escape or quote the percent sign (`\%` or `"%"`).

Note: The shell runs only the first line of the command field (up to a `%` or end of line). All other lines are made available to the command as standard input.

The **cron** command invokes a subshell from your **\$HOME** directory. This means that it will not run your **.profile** file. If you schedule a command to run when you are not logged in and you want to have commands in your **.profile** run, you must explicitly do so in the **crontab** file. (For a more detailed discussion of how **sh** can be invoked, see “**sh**” on page 913).

cron supplies a default environment for every shell, defining **HOME**, **LOGNAME**, **SHELL** (`= /bin/sh`), and **PATH** (`= :/bin:/usr/bin`).

Flags

- l** Lists your **crontab** file.
- r** Removes your **crontab** file from the **crontab** directory.

crontab

Examples

The following examples show valid **crontab** file entries.

1. To write the time to the console every hour on the hour:

```
0 * * * * echo The hour is `date`. >/dev/console
```

This example uses **command substitution**. For more information, see “Command Substitution” on page 925.

2. To run **calendar** at 6:30 a.m. every Monday, Wednesday, and Friday:

```
30 6 * * 1,3,5 /usr/bin/calendar -
```

3. To define text for the standard input to a command:

```
0 16 10-31 12 5 /etc/wall%HAPPY HOLIDAYS!%Remember to turn in your time card.
```

This writes a message to all users logged in at 4:00 p.m. each Friday between December 10th and 31st.

The text following the % (percent sign) defines the standard input to the **wall** command as:

```
HAPPY HOLIDAYS!  
Remember to turn in your time card.
```

Files

/usr/lib/cron	Main cron directory.
/usr/spool/cron/crontabs	Spool area.
/usr/lib/cron/cron.allow	List of allowed users.
/usr/lib/cron/cron.deny	List of denied users.

Related Information

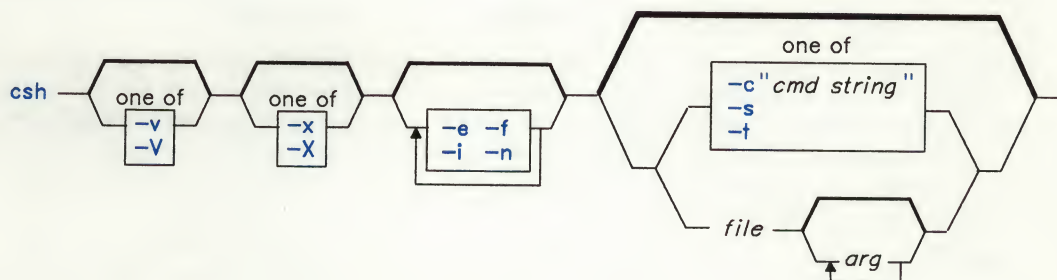
The following commands: “**cron**” on page 220 and “**sh**” on page 913.

csh

Purpose

Interprets commands read from a file or entered from the keyboard.

Syntax



OL805447

Description

The **csh** command is a system command interpreter and programming language that incorporates a history mechanism and a C-like syntax. Like the **sh** command, it is an ordinary user program that reads commands typed at the keyboard and arranges for their execution. In addition, it can read commands from a file, usually called a *shell procedure* or a *command file*.

When you run **csh**, it begins by executing commands from the file **.cshrc** in your home directory, if it exists. If, on the other hand, **csh** runs as a login shell, it executes commands from your **.cshrc** file and your **.login** file.

Commands

A *simple command* is a sequence of *words* separated by single blanks or tabs.

Japanese Language Support Information

Words can also be separated by double blanks.

A word is a sequence of characters and/or numerals that does not contain blanks without quotation marks. In addition, the following characters and doubled characters also form single words when used as command separators or terminators:

```
&    |    ;    <    >    (    )
&&   ||   <<   >>
```

These special characters may be parts of other words. Preceding them with a \ (backslash), however, prevents the shell from interpreting them as special characters. When the shell is not reading input from a work station, it treats any word that begins with a # (number sign) as a comment and ignores that word and all characters following up to the next new-line character. Strings enclosed in ' ' or " " (matched pairs of quotation characters) or ` ` (grave accents) can also form parts of words. (Blanks, tab characters, and special characters do not form separate words when they are found within these quotation marks.) In addition, within ' ' or " " (pairs of single or double quotation marks), you may include the new-line character by preceding it with \ (backslash).

The first word in the simple-command sequence (numbered 0), usually specifies the name of a command. Any remaining words, with a few exceptions, are passed to that command. If the command specifies an executable file that is a compiled program, the shell immediately runs that program. If the file is marked executable but is not a compiled program, the shell assumes that it is a shell procedure. In this case it spawns another instance of itself (a **subshell**), to read the file and execute the commands included in it.

A **pipeline** is a sequence of one or more commands separated by a | (vertical bar). The output of each command in a pipeline provides the input to the next command.

A **list** is a sequence of one or more pipelines separated by a ; (semicolon), & (ampersand), && (two ampersands), or || (two vertical bars) and optionally ended by a ; (semicolon) or an & (ampersand). These separators and terminators have the following effects:

- ;
Causes **sequential execution** of the preceding pipeline (the shell waits for the pipeline to finish).
- &
Causes **asynchronous execution** of the preceding pipeline (the shell does *not* wait for the pipeline to finish).
- &&
Causes the list following it to be executed *only* if the preceding pipeline returns a zero exit value.
- ||
Causes the list following it to be executed *only* if the preceding pipeline returns a nonzero exit value.

Note: The **cd** command is an exception. If it returns a nonzero exit value, no subsequent commands in a list are executed, regardless of the separators.

The ; and & separators have equal precedence, as do && and ||. The single-character separators have lower precedence than the double-character separators. A new-line character without quotation marks following a pipeline functions the same as a ; (semicolon). Place any of the above in parentheses to form a simple command.

The shell associates a **job** with each pipeline. It keeps a table of current jobs and assigns them small integer numbers. When you start a job asynchronously by terminating the command with a **&**, the shell displays a line that looks like the following:

```
[1] 1234
```

This line indicates that the job number is 1 and that the job is composed of one process with a process-ID of 1234. Use the built-in **jobs** command (page 243) to see what jobs are currently running.

A job running in the background competes for input if it tries to read from the work station. Background jobs can also produce output that competes for the work station and is interleaved there with the output of other jobs.

There are several ways to refer to jobs in the shell. Use the **%** (percent) character to introduce a job name. This name can be either the job number or the command name that started the job, if this name is unique. So, for example, if a **make** process is running as job 1, you can refer to it as **%1**. You can also refer to it as **%make**, if there is only one suspended job with a name that begins with the string **make**. You can also use

?:string

to specify a job whose name contains *string*, if there is only one such job.

The shell detects immediately whenever a process changes state. Whenever a job becomes blocked so that further progress is not possible, a message is sent to the work station, but not until just before the shell prompt. If, however, the **notify** shell variable is set (see page 237), the shell issues a message that indicates changes in status of background jobs immediately. Use the **notify** built-in command (page 244) to mark a single process so that its status changes are immediately reported. By default, **notify** marks the current process.

History Substitution

History substitution lets you modify individual words from previous commands to create new commands, thus making it easy to repeat commands, repeat the arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing.

History substitutions begin with the **!** (exclamation) character and may appear anywhere on the command line, provided they do not nest (in other words, a history substitution cannot contain another history substitution). You can precede the **!** with a **** to prevent its special meaning. In addition, if you place the **!** before a blank, tab, new-line character, **=** (equal sign), or **(** (left parenthesis), it is passed unchanged. History substitutions also occur when you begin an input line with a **^** (circumflex). (This special abbreviation is discussed on page 230.) The shell echoes any input line containing history substitutions at the work station before it executes that line.

The history list saves commands that the shell reads from the work station and that consist of one or more words. History substitution reintroduces sequences of words from these saved commands into the input stream.

The **history** shell variable (page 237) controls the size of the history list. You must set the **history** shell variable either in the **.cshrc** file or on the command line with the built-in **set** command (page 245). The previous command is always retained, however, regardless of the value of **history**. Commands in the history list are numbered sequentially starting from 1. The built-in **history** command (page 242) produces output of the type:

```

9  write michael
10 ed write.c
11 cat oldwrite.c
12 diff *write.c

```

The command strings are shown with their event numbers. It is not usually necessary to use event numbers to refer to events, but you can have the current event number displayed as part of your system prompt by placing an **!** in the prompt string assigned to the **prompt** environmental variable (page 238).

A full history reference contains an event specification, a word designator, and one or more modifiers in the following general format:

event[:word:modifier[:modifier]] . . .

Note: Only one word can be modified. A string that contains blanks is not allowed.

In the previous sample of **history** command output, the current event number is 13. Using this example, the following refer to previous events:

Event Specification

!10	Refers to event number 10
!-2	Refers to event number 11 (the current event minus 2)
!d	Refers to a command word beginning with d (in this case event number 12)
!?mic?	Refers to a command word that contains the string mic (in this case, event number 9).

These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case, **!!** refers to the previous command; the command **!!** alone on an input line reruns the previous command.

To select words from an event, follow the event specification with a **:** (colon) and one of the following word designators (the words of an input line are numbered sequentially starting from 0):

Word Designator

0	The first word (the command name)
<i>n</i>	The <i>n</i> th argument
^	The first argument
\$	The last argument
%	The word matched by an immediately preceding <i>?string?</i> search
<i>x-y</i>	A range of words from the <i>x</i> th word to the <i>y</i> th word
- <i>y</i>	A range of words from the first word (0) to the <i>y</i> th word
*	The first through the last argument, or nothing if there is only one word (the command name) in the event
<i>x</i> *	The <i>x</i> th through the last argument
<i>x-</i>	Like <i>x</i> * but omitting the last word.

You may omit the colon that separates the event specification from the word designator if the word designator begins with a ^, \$, *, -, or %. You can also place a sequence of the following modifiers after the optional word designator, each preceded by a colon:

Modifier

h	Remove a trailing path name extension, leaving the head.
r	Remove a trailing ".xxx" component, leaving the root name.
e	Remove all but the trailing extension ".xxx."
s/<i>l</i>/<i>r</i>/	Substitute <i>l</i> for <i>r</i> . With substitutions, it is an error for no word to be applicable.

The *l* (left) side of a substitution is not a pattern in the sense of a string recognized by an editor; rather, it is a word, a single unit without blanks. Normally, a / (slash) delimits the word (*l*) and its replacement (*r*). However, you can use any character as the delimiter if you precede that character with a \ (backslash). Thus, in the following example:

```
s%/usr/myfile\%/usr/yourfile\%
```

the % becomes the delimiter allowing you to include the / in your word. If you include an & in the replacement, it is replaced by the text from the left-hand side (*l*). A null *l* side is replaced by either the last substitution or by the last string used in the contextual scan *!?string?*. You may omit the trailing delimiter (/) if a new-line character follows immediately.

t	Remove all leading path name components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change globally, that is, g& .
p	Display the new command, but do not run it.
q	Quote the substituted words, thus preventing further substitutions.
x	Act like q , but break into words at blanks, tabs, and new-line characters.

Unless the modifier is preceded by a **g**, the change applies only to the first modifiable word.

If you give a history reference without an event specification, for example, **!\$**, the shell uses the previous command as the event, unless a previous history reference occurs on the same line, in which case it repeats the previous reference. Thus, the following sequence:

```
!?foo?^ !$
```

gives the first and last arguments of the command that matches **?foo?**.

A special abbreviation of a history reference occurs when the first non-blank character of an input line is a **^** (circumflex). This is equivalent to **!:s^**, thus providing a convenient shorthand for substitutions on the text of the previous line. The command **^lb^lib** corrects the spelling of **lib** in the previous command.

You can enclose a history substitution in **{ }** (braces), if necessary, to insulate it from the characters that follow. For example, if you want to use a reference to the command:

```
ls -ld ~paul
```

to perform the command:

```
ls -ld ~paula
```

use the following:

```
!{1}a
```

whereas **!la** would look for a command starting with **la**.

Quoting with Single and Double Quotes

Enclose strings in single and double quotation marks to prevent all or some of the substitutions that remain. Enclosing strings in **' '** (single quotation marks) prevents any further interpretation. Enclosing strings in **" "** (double quotation marks) allows further expansion. In both cases, the text that results becomes (all or part of) a single word. Only in one special case does a string quoted by **" "** yield parts of more than one word; strings quoted by **' '** never do (see "Command Substitution" on page 231).

Command and File-Name Substitution

The shell performs command and file-name substitutions selectively on the arguments of built-in commands. This means that it does not expand those parts of expressions that are not evaluated. For commands that are not built-in, the shell substitutes the command name separately from the argument list. This occurs very late, after it performs input/output redirection and in a child of the main shell.

Command Substitution

The shell performs command substitution on a command string enclosed in `` (grave accents). The shell normally breaks the output from such a command into separate words at blanks, tabs, and new-line characters; this text then replaces the original command string. Within strings surrounded by " " (double quotation marks), the shell treats only the new-line character as a word separator, thus preserving blanks and tabs within the word.

In any case, the single final new-line character does not force a new word. Note that it is therefore possible for command substitution to yield only part of a word, even if the command outputs a complete line.

File-name Substitution

If a word contains any of the characters *, ?, [, or {, or begins with the ~ character, that word is a candidate for file-name substitution, also known as **globbing**. The word is then regarded as a pattern and replaced with an alphabetically sorted list of file names which match the pattern.

The current collating sequence is used, which may be specified by the environment variables **NLCTAB** or **NLFILE**. In a list of words specifying file-name substitution, it is an error for no patterns to match an existing file name, but it is not required that each pattern match. Only the character-matching symbols *, ?, and [imply pattern matching; the characters ~ and { being more related to abbreviations.

In matching file names, the character . (dot) at the beginning of a file name or immediately following a /, and the character /, must be matched explicitly. The * character matches any string of characters, including the null string. The ? character matches any single character. The sequence [abcd] matches any one of the enclosed characters. Within [], a lexical range of characters may be indicated by [a-z]. The characters that match this pattern are defined by the current collating sequence (see "ctab" on page 257).

Japanese Language Support Information

Note: For information about matching file names, see “File-name Substitution in Japanese Language Support” on page 233.

The ~ (tilde) character at the beginning of a file name is used to see home directories. Standing alone, ~ expands to your home directory as reflected in the value of the home shell variable. When followed by a name that consists of letters, digits, and - (dash) characters, the shell searches for a user with that name and substitutes their home directory. Thus, ~ken might expand to /usr/ken and ~ken/chmach to /usr/ken/chmach. If the ~ character is followed by a character other than a letter or /, or appears anywhere except at the beginning of a word, it is left undisturbed.

The pattern a{b,c,d}e is a shorthand for abe ace ade. The shell preserves the left-to-right order, with results of matches being stored separately at a low level to preserve this order. This construct may be nested. Thus:

```
~source/s1/{oldls,ls}.c
```

expands to:

```
/usr/source/s1/oldls.c /usr/source/s1/ls.c
```

if the home directory for source is /usr/source. Similarly:

```
../{memo,*box}
```

might expand to:

```
../memo ../box ../mbox
```

(Note that memo is not sorted with the results of matching *box.) As a special case, {, }, and {} are passed undisturbed.

File-name Substitution in Japanese Language Support

You can also use the following notation to match file names within a range indication:

`[:charclass:]`

This format instructs the system to match any single character belonging to *class*; the defined classes correspond to **c**type subroutines. Following are the names of these classes:

alnum	jalpha
alpha	jdigit
digit	jhira
lower	jkanji
print	jkata
punct	jparen
space	jpunct
upper	jspace
xdigit	jxdigit

For example, the expression that matches any single kanji character would be the following:

`[[:jkanji:]]`

For additional information about character class expressions, see the discussion of this topic in “**ed**” on page 371.

Alias Substitution

The shell maintains a list of aliases that the **alias** and **unalias** built-in commands (page 240) can establish, display, and modify. After the shell scans the command line, it divides it into distinct commands and checks the first word of each command, left to right, to see if it has an alias. If it does, the shell uses the history mechanism available (see “History Substitution” on page 227), to replace the text of the alias with the text of the command it stands for. The words that result replace the command and argument list. If reference is not made to the history list, the argument list is left unchanged. Thus, if the alias for the **ls** command is **ls -l**, the shell replaces the command **ls /usr** with **ls -l /usr**. The argument list is undisturbed because there is no reference to the history list in the command with an alias. Similarly, if the alias for **lookup** is:

`grep \!^ /etc/passwd`

then the shell replaces **lookup bill** with:

`grep bill /etc/passwd`

Here, `!` refers to the history list and the shell replaces it with the first argument in the input line, in this case `bill`. Note that you can use special pattern-matching characters in an alias. Thus the command:

```
alias lprint 'pr \!* >> print'
```

makes a command which formats its arguments to the line printer. The `!` is protected from the shell in the alias so that it is not expanded until `pr` runs.

If an alias is found, the word transformation of the input text is performed and the alias process begins again on the reformed input line. If the first word of the next text is the same as the old, looping is prevented by flagging it to terminate the alias process. Other loops are detected and cause an error.

Variable Substitution

The shell maintains a set of variables, each of which has as its value a list of zero or more words. Some of these variables are set by the shell or referred to by it. For instance, the `argv` variable is an image of the shell variable list, and words which comprise the value of this variable are referred to in special ways.

You can change and display the values of variables with the `set` and `unset` commands. Of the variables referred to by the shell, a number are toggles (variables that turn something on and off); the shell does not care what their value is, only whether they are set or unset. For instance, the `verbose` variable is a toggle which causes command input to be echoed. The setting of this variable results from the `-v` flag on the command line.

Other operations treat variables numerically. The `@` command performs numeric calculations and the result is assigned to a variable. Variable values are, however, always represented as (zero or more) strings. For numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After an input line is parsed and alias substitution is performed, and before each command is run, variable substitution is performed, keyed by `$` characters. You can prevent this expansion by preceding the `$` with a `\`, except within `" "` (double quotation marks, where it always occurs, and within `' '` (single quotation marks), where it never occurs. Strings quoted by `' '` are interpreted later (see "Command Substitution" on page 231), so `$` substitution does not occur there until later, if at all. A `$` is passed unchanged if it is followed by a blank, tab, or new-line character.

Input/output redirections are recognized before variable expansion and are expanded separately. Otherwise, the command name and complete argument list expands together. It is therefore possible for the first (command) word to this point to generate more than one word, the first of which becomes the command name and the rest of which become parameters.

Unless enclosed in " " or given the **:q** modifier, the results of variable substitution may themselves eventually be command and file name substituted. Within pairs of double quotation marks, a variable with a value that consists of multiple words expands to a (portion of a) single word, with the words of the variable's value separated by blanks. When you apply the **:q** modifier to a substitution, the variable expands to multiple words. Each word is separated by a blank and quoted to prevent later command or file name substitution.

The following notation allows you to introduce variable values into the shell input. Except as noted, it is an error to reference a variable that is not set.

\$name
\${name}

Replaced by the words assigned to *name*, each separated by a blank. Braces insulate *name* from any following characters that would otherwise be part of it. Shell variable names start with a letter and consist of up to 20 letters and digits, including the **_** (underline) character. If *name* is not a shell variable but is set in the environment, then that value is returned. The **:** modifiers and the other forms given below are not available in this case.

\$name[selector]
\${name[selector]}

Used to select only some of the words from the value of *name*. The selector is subjected to **\$** substitution and may consist of a single number, or two numbers separated by a **-**. The first word of a variable's string value is numbered 1. If the first number of a range is omitted, it defaults to 1. If the last member of a range is omitted, it defaults to **\$#name**. The ***** symbol selects all words. It is not an error for a range to be empty if the second argument is omitted or is in range.

\$#name
 \${#name}

Gives the number of words in the variable. This is useful for later use in a **[selector]**.

\$0

Substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

\$number
\${number}

Equivalent to **\$argv[number]**

\$*

Equivalent to **\$argv[*]**.

You can apply the modifiers **:gh**, **:gt**, **:gr**, **:h**, **:r**, **:q**, and **:x** to the substitutions above. If **{}** (braces) appear in the command form, then the modifiers must appear within the braces. The current implementation allows only one **:** modifier on each **\$** expansion.

The following substitutions may not be changed with **:** modifiers.

`\${name}`	Substitutes the string 1 if name is set; 0 if it is not set.
`\$?`	Substitutes 1 if the current input file name is known; 0 if it is not known.
`\$`	Substitutes the (decimal) process number of the (parent) shell.
`\$<`	Substitutes a line from the standard input, without further interpretation. Use it to read from the keyboard in a shell procedure.

Predefined and Environmental Variables

The following variables have special meaning to the shell. Of these, **argv**, **cwd**, **home**, **path**, **prompt**, **shell**, and **status** are always set by the shell. Except for **cwd** and **status**, this setting occurs only at initialization. These variables maintain their settings unless you explicitly reset them.

The **cs**h command copies the environment variables **USER**, **TERM**, **HOME**, and **PATH** into the **cs**h variables **user**, **term**, **home**, and **path**, respectively. The values are copied back into the environment whenever the normal shell variables reset. It is not necessary to worry about the setting of the **path** variable other than in the **.cshrc** file, since **cs**h subprocesses import the definition of **path** from the environment and re-export it if it is changed.

argv	Set to the arguments to the shell; it is from this variable that positional parameters are substituted.
cdpath	Can be given a list of alternate directories to be searched by the chdir commands to find subdirectories.
cwd	The full path name of the current directory.
echo	Set when the -x command line flag is used; when set, causes each command and its arguments to echo just before it is run. For non-built-in commands, all expansions occur before echoing. Built-in commands are echoed before command and file name substitution, since these substitutions are then done selectively.
histchars	Can be given a string value to change the characters used in history substitution. Use the first character of its value as the history substitution character, this replaces the default character ! . The second character of its value replaces the ^ (circumflex) character in quick substitutions.

history	Can be given a numeric value to control the size of the history list. Any command that is referenced in this many events is not discarded. Very large values of history may run the shell out of memory. Saves the last command that ran on the history list, regardless of whether history is set.
home	Your home directory, initialized from the environment. The file name expansion of ~ refers to this variable.
ignoreeof	If set, the shell ignores an end-of-file character from input devices that are work stations. This prevents shells from accidentally being killed when it reads an end-of-file character (Ctrl-D).
mail	<p>The files where the shell checks for mail. This is done after each command completion, which results in a prompt if a specified interval has elapsed. The shell displays the message, "You have new mail" if the file exists with an access time not greater than its change time.</p> <p>If the first word of the value of mail is numeric, it specifies a different mail checking interval (in seconds); the default is 10 minutes. If you specify multiple mail files, the shell displays the message, "New mail in <i>file</i>", when there is mail in <i>file</i>.</p>
noclobber	If set, places restrictions on output redirection to insure that files are not accidentally destroyed, and that > > redirections see existing files. (See "Redirecting Input and Output" on page 238).
noglob	If set, inhibits file-name expansion. This is most useful in shell procedures that are not dealing with file names, or after a list of file names has been obtained and further expansions are not desirable.
nonomatch	If set, it is not an error for a file-name expansion to not match any existing files; rather, the primitive pattern returns. It is still an error for the primitive pattern to be malformed.
notify	If set, the shell notifies asynchronously of changes in job status. The default presents status changes just before displaying the shell prompt.

path	Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no path variable set, then only full path names run. The usual search path is the current directory, /bin , and /usr/bin . For the superuser, the default search path is /etc , /bin , and /usr/bin . A shell which is given neither the -c nor the -t flags normally hashes the contents of the directories in the path variable after reading .cshrc and each time the path variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the rehash command (page 245), or the commands may not be found.
prompt	The string which is displayed before each command is read from an interactive work station input. If a ! appears in the string, it is replaced by the current event number. If the ! is in a quoted string, it must be preceded by a \ (backslash). The default prompt is % , # for the superuser.
savehist	Given a numeric value to control the number of entries of the history list that are saved in ~/.history when you log off. Any command which is referenced in this many events is saved. During startup, the shell reads ~/.history into the history list, enabling history to be saved across logins. Very large values of savehist slow down the shell startup.
shell	The file in which the shell resides. This is used in forking shells to interpret files which have execute bits set, but which are not executable by the system (see "Nonbuilt-in Command Execution" on page 249). This is initialized to the (system-dependent) home of the shell.
status	The status returned by the last command. If it ended abnormally, then 0200 is added to the status. Built-in commands that fail return exit status 1; all other built-in commands set status 0.
time	Controls automatic timing of commands. If set, any command that takes more than the specified number of CPU seconds causes a line giving user, system, and real times and a utilization percentage, that is the ratio of user-plus-system-times to real time, displays when it ends.
verbose	Set by the -v command line flag; causes the words of each command to display after history substitution.

Redirecting Input and Output

You can redirect the standard input and standard output of a command with the following syntax:

< name Opens file *name* (which is first variable, command, and file name expanded) as the standard input.

<< *word* Reads the shell input up to a line which is the same as *word*. *word* is not subjected to variable, file name, or command substitution, and each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting character (\, ", ', or `) appears in *word*, the shell performs variable and command substitution on the intervening lines, allowing \ to quote \$, \, and `. Commands which are substituted have all blanks, tabs, and new-line characters preserved, except for the final new-line character, which is dropped. The resultant text is placed in an anonymous temporary file, which is given to the command as standard input.

> *name*

>! *name*

>& *name*

>&! *name*

Uses the file *name* as standard output. If the file does not exist, it is made. If the file exists, it is truncated, its previous contents being lost. If the **noclobber** shell variable is set, the file must not exist or be a character special file, or an error results. This helps prevent accidental destruction of files. In this case, use the ! forms to suppress this check. The forms involving & route the diagnostic output into the specified file as well as the standard output. *name* expands in the same way as < input file names.

>> *name*

>>& *name*

>>! *name*

>>&! *name*

Uses file *name* as standard output like >, but places output at the end of the file. If the **noclobber** shell variable is set, it is an error for the file not to exist, unless one of the ! forms is given. Otherwise, it is similar to >.

A command receives the environment in which the shell was invoked, as changed by the input/output parameters and the presence of the command as a pipeline. Thus, unlike some previous shells, commands that run from a file of shell commands do not have any access to the text of the commands by default. Rather, they receive the original standard input of the shell. Use the << mechanism to present in-line data. This lets shell command files function as components of pipelines and lets the shell block read its input. Note that the default standard input for a command run detached is not changed to be the empty file /dev/null. Rather, the standard input remains as the original standard input of the shell.

To redirect the diagnostics output through a pipe with the standard output, use the form !& (vertical bar ampersand) rather than just ! (vertical bar).

Control Flow

The shell contains some commands that can be used to regulate the flow of control in command files (shell procedures) and (in limited but useful ways) from work station input. These commands all operate by forcing the shell to reread or skip in its input and, because of the implementation, restrict the placement of some of the commands.

The **foreach**, **switch**, and **while** statements, and the **if-then-else** form of the **if** statement, require that the major keywords appear in a single simple command on an input line.

If the shell input is not searchable, the shell buffers input whenever a loop is being read and searches the internal buffer to do the rereading implied by the loop. To the extent that this allows, backward **gotos** succeed on inputs that you cannot search.

Built-in Commands

Built-in commands are run within the shell. If a built-in command occurs as any component of a pipeline except the last, it runs in a subshell.

Notes:

1. If you enter a command from **cs** at the prompt, the system searches for a **cs** built-in command first. If a built-in command does not exist, then the system searches for an AIX command. Some **cs** built-in commands and AIX commands have the same name. However, these commands do not necessarily work the same way. Check the appropriate command description for information on how the command works.
2. If you run a shell procedure from **cs** and the first characters of the shell procedure are `#!/shell_pathname`, **cs** runs the shell specified in the comment to process the procedure. Otherwise, **cs** runs the standard shell (**sh**). If run by **sh**, **cs** built-in commands are not recognized. To get the system to run **cs** commands, the first line of the procedure should be: `#!/bin/cs`

alias

alias name

alias name wordlist

Displays all aliases (first form). The second form displays the alias for *name*. The final form assigns the specified *wordlist* as the alias of *name*. *wordlist* is command and file name substituted. *name* is not allowed to be **alias** or **unalias**.

break

Resumes running after the end of the nearest enclosing **foreach** or **while**. Runs the remaining commands on the current line. Multilevel breaks are therefore possible by writing them all on one line.

breaksw

Breaks from a **switch**; resumes after the **endsw**.

case label:

Defines a label in a **switch** statement, as discussed in the following.

cd
cd *name*
chdir
chdir *name*

Changes the current directory to *name*. If no argument is given, then changes to your home directory.

If *name* is not found as a subdirectory of the current directory and does not begin with /, ./, or ../, then each component of the **cdpath** shell variable is checked to see if it has a subdirectory *name*. Finally, if all else fails, but *name* is a shell variable with a value that begins with /, then this is tried to see if it is a directory.

continue

Continues execution of the nearest enclosing **while** or **foreach**. The rest of the commands on the current line run.

default:

Labels the default case in a **switch** statement. The default should come after all **case** labels.

dirs

Displays the directory stack, the top of the stack is at the left, the first directory in the stack being the current directory.

dirstyle +
dirstyle -

Interprets directories in specified format so you can read their contents in System V format. The + flag converts path names from remote file systems to System V format. The - flag leaves the directory contents in raw form instead of converting the path names of remote file systems to the System V format.

echo *string*
echo -n *string* . . .

Writes the listed *strings* to the shell's standard output, separated by spaces and ending with a new-line character unless you specify the -n flag.

else
end
endif
endsw

See the description of the **foreach**, **if**, **switch**, and **while** statements.

eval *arg* . . .

Reads *arg* as input to the shell and runs the resulting command(s) in the context of the current shell. Use this to run commands generated as the result of command or variable substitution, since parsing occurs before these substitutions.

exec *cmd*
exit

Runs the specified command in place of the current shell.

exit (<i>expr</i>)	Exits the shell with either the value of the status shell variable (first form) or with the value of the specified expression (second form).
foreach <i>name</i> (<i>list</i>) end	Successively sets <i>name</i> to each member of <i>list</i> and runs the sequence of commands between the foreach and the matching end . Both foreach and end must appear alone on separate lines. Use the continue statement to continue the loop and the break statement to end the loop prematurely. When this command is read from the work station, the loop is read once, prompts with ? before any statement in the loop runs. If a mistake is made in entering a loop, it can be corrected before you run the loop. Commands within loops, prompted for by ?, are not placed in the history list.
glob <i>list</i>	Functions like echo , but does not recognize backslash (\) escapes, and delimits words by null characters in the output. Useful if you want to use the shell to perform file-name substitution to expand a list of words.
goto <i>word</i>	Continues to run after the line specified by <i>word</i> . The specified <i>word</i> is file-name and command expanded to yield a string of the form <i>label</i> . The shell rewinds its input as much as possible and searches for a line of the form <i>label</i> :, possibly preceded by blanks or tabs.
history history <i>num</i> history -r <i>num</i> history -h <i>num</i>	Displays the history event list. If you specify a number, only the <i>n</i> most recent events are displayed. The -r flag reverses the order of display to the most recent first rather than the oldest first. The -h flag causes the history list to be displayed without leading numbers. Use this to produce files suitable for used with the -h flag of the source command.
if (<i>expr</i>) <i>cmd</i>	Runs the single command (with arguments) if the specified expression evaluates true. Variable substitution on <i>cmd</i> happens early, at the same time it does for the rest of the if statement. <i>cmd</i> must be a simple command, not a pipeline, command list, or parenthesized command list. Note: Input and output redirection occurs even if <i>expr</i> is false (and the command is not executed).
if (<i>expr</i>) then	

...
else if (*expr2*) then

...
else

...
endif

If *expr* is true, runs the commands that follow the first **then**; **else if** *expr2* is true, runs the commands that follow the second **then**; **else** runs the commands that follow the second **else**. Any number of **else-if** pairs are possible; only one **endif** is needed. The **else** part is optional. The words **else** and **endif** must appear at the beginning of input lines. The **if** must appear alone on its input line or after an **else**.

jobs
jobs -l

Lists the active jobs. With the **-l** flag, lists process-IDs in addition to the job number and process-ID.

kill %*job*
kill -*signal* %*job* . . .
kill *pid*
kill -*signal* *pid* . . .
kill -l

Sends to the jobs or process that you specify either the **TERM** (terminate) signal or *signal*. Specify *signals* either by number or by names (as given in `/usr/include/signal.h`, stripped of the SIG prefix). Signal names are listed by **kill -l**.

limit
limit *resource*
limit *resource max-use*

Limits the usage by the current process and each process it creates to not individually exceed *max-use* on the specified *resource*. If a *max-use* is not given, the current limit displays; if a *resource* is not given, all limitations are given. Controllable resources are limited to **filesize**, **stacksize**, and **datasize**. You can specify *max-use* as a (floating-point or integer) number followed by a scale factor: **k** or **kilobytes** (1024 bytes), **m** or **megabytes**, or **b** or **blocks** (the units used by the **ulimit** system call). For both *resource* names and scale factors, unambiguous prefixes of the names suffice. The **filesize** may be lowered by an instance of **csch**, but may only be raised by an instance whose effective user-ID is **root**. (See the **ulimit** system call in *AIX Operating System Technical Reference*.)

login

Ends a login shell, and replaces it with an instance of `/bin/login`. This is one way to log off (included for compatibility with the **sh** command).

logout

Ends a login shell. Especially useful if **ignoreeof** is set.

newgrp

Executes the **newgrp** command in the current shell process. See “**newgrp**” on page 689 for a discussion of command options.

nice

nice + *num*

nice *cmd*

nice + *num cmd*

Sets the priority of commands run in this shell to 24 (first form). The second form sets the priority to the specified number. The final two forms run the specified command at priority 24 and the specified number, respectively. If you have superuser authority, you can specify **nice** with a negative number. The command always runs in a subshell, and the restrictions placed on commands in simple **if** statements apply.

nohup

nohup *cmd*

Causes hangups to be ignored for the remainder of the procedure (first form). The second form causes the specified command to be run with hangups ignored. To run a pipeline or list of commands with this form, put the pipeline or list in a shell procedure, give the procedure execute permission, and use the shell procedure as the *cmd*. All processes run in the background with **&** are effectively protected from being sent a hangup signal when you log off, but will still be subject to explicitly sent hangups unless **nohup** is used.

notify

notify %*job* . . .

Causes the shell to notify you asynchronously when the status of the current or specified jobs changes. Normally, notification is presented just before the shell prompt. This is automatic if the **notify** shell variable is set.

onintr

onintr -

onintr *label*

Controls the action of the shell on interrupts. The first form restores the default action of the shell on interrupts, which is to end shell procedures or to return to the work station command input level. The second form causes all interrupts to be ignored. The third form causes the shell to run a **goto** *label* when it receives an interrupt or a child process ends due to an interruption. In any case, if the shell is running detached and interrupts are being ignored, all forms of **onintr** have no meaning, and interrupts continue to be ignored by the shell and all invoked commands.

popd

popd + *n*

Pops the directory stack, returns to the new top directory. With a + *n*, discards the *n*th entry in the stack. The elements of the directory stack are numbered from the top starting at 0.

pushd
pushd *name*
pushd *+n*

With no arguments, exchanges the top two elements of the directory stack. With *name*, changes to the new directory and pushes the old current directory (as given in the **cwd** shell variable) onto the directory stack. With a numeric argument, rotates the *n*th argument of the directory stack around to be the top element and changes to it. The members of the directory stack are numbered from the top starting at 0.

rehash

Causes the internal hash table of the contents of the directories in the **path** shell variable to be recomputed. This is needed if new commands are added to directories in **path** while you are logged in. This should only be necessary if commands are added to one of the user's own directories, or if someone changes the contents of one of the system directories.

repeat *count cmd*

Runs the specified command, which is subject to the same restrictions as the **if** statement, *count* times.

Note: I/O redirections occur exactly once, even if *count* is 0.

set
set *name*
set *name=word*
set *name[index]=word*
set *name=(list)*

Shows the value of all shell variables (first form). Variables that have more than a single word as their value are displayed as a parenthesized word list. The second form sets *name* to the null string. The third form sets the *index*th component of *name* to *word*; this component must already exist. The final form sets *name* to the list of words in *list*. In all cases, the value is command and file-name expanded. These arguments may be repeated to set multiple values in a single **set** command. However, variable expansion happens for all arguments before any setting occurs.

setenv *name value*

Sets the value of environment variable *name* to be *value*, a single string. The most commonly used environment variables, **USER**, **TERM**, and **PATH**, are automatically imported to and exported from the **csH** variables **user**, **term**, and **path**; there is no need to use **setenv** for these.

If you modify the environment variables **NLFILE** or **NLCTAB**, the current international character support environment and collating sequence are changed as specified for subsequent commands executed from the shell.

shift

shift *variable*

Shifts the members of **argv** to the left. It is an error for **argv** not to be set or to have less than one word as its value. The second form does the same function on the specified *variable*.

source *name*

source -h *name*

Reads commands from *name*. You can nest the **source** commands. However, if they are nested too deeply, the shell may run out of file descriptors. An error in a **source** command at any level ends all nested **source** commands. Normally, input during **source** commands is not placed on the history list. The **-h** flag causes the commands to be placed in the history list without running.

switch (*string*)

case *str1*:

...

breaksw

default:

...

breaksw

endsw

Successively matches each case label against *string*. The *string* is command and file-name expanded first. Use the pattern-matching characters *****, **?**, and **[. . .]** in the case labels, which are variable expanded. If none of the labels match before a **default** label is found, then the execution begins after the **default** label. Each **case** label and the **default** label must appear at the beginning of a line. The **breaksw** command causes execution to continue after the **endsw**. Otherwise, control may fall through case labels and the **default** labels, as in C. If no label matches and there is no **default**, execution continues after the **endsw**.

time

time *cmd*

With no argument, displays a summary of time used by this shell and its children. If arguments are given, the specified command is timed, and a time summary as described under the **time** shell variable is displayed. If necessary, an extra shell is created to display the time statistic when the command completes.

umask

umask *value*

Displays the file creation mask (first form) or sets it to the specified *value* (second form). The mask is given as an octal value. Common values for the mask are 002, giving all access to owner and group and read and execute access to others, or 022, giving all access to the owner and all access except write access for users in the group or others.

unalias *pattern*

Discards all aliases with names that match *pattern*. Thus, all aliases are removed by **unalias ***. The absence of aliases does not cause an error.

unhash	Disables the use of the internal hash table to locate running programs.
unlimit unlimit <i>resource</i>	Removes the limitation on <i>resource</i> . If you do not specify <i>resource</i> , then all <i>resource</i> limitations are removed. The only removable limitation is that on filesize , and only the superuser can remove it.
unset <i>pattern</i>	Removes all variables with names that match the <i>pattern</i> . Use unset * to remove all variables. It is not an error for nothing to be unset.
unsetenv <i>pattern</i>	Removes all variables from the environment whose names match the specified <i>pattern</i> . (See the setenv built-in command on page 245.)
wait	Waits for all background jobs. If the shell is interactive, an INTERRUPT (Alt-Pause) can disrupt the wait, when the shell displays the names and job numbers of all jobs known to be outstanding.
while (<i>expr</i>) ... end	Evaluates the commands between the while and the matching end while <i>expr</i> evaluates nonzero. You can use break to end and continue to continue the loop prematurely. The while and end must appear alone on their input lines. If the input is a work station, prompts occur the first time through the loop, as for the foreach statement.
@ @ <i>name</i> = <i>expr</i> @ <i>name</i> [<i>index</i>] = <i>expr</i>	Displays the values of all the shell variables (first form). The second form sets the specified <i>name</i> to the value of <i>expr</i> . If the expression contains <, >, &, or !, then at least this part of the expression must be placed within parentheses. The third form assigns the value of <i>expr</i> to the <i>index</i> th argument of <i>name</i> . Both <i>name</i> and its <i>index</i> th component must already exist. C operators, such as *= and += are available. The space separating <i>name</i> from the assignment operator is optional. Spaces are, however, required in separating components of <i>expr</i> , which would otherwise be single words. Special postfix ++ and -- operators increase and decrease <i>name</i> .

Expressions

The **@** built-in command and the **exit**, **if**, and **while** statements accept expressions which include operators similar to those of C, with the same precedence. The following operators are available:

```
*      /      %
+      -
<<    >>
<=    >=    >
==     !=     =~    !~
```

In the preceding list, operators of equal precedence appear on the same line, below those lines containing operators (if any) that have greater precedence and above those lines containing operators having lesser precedence. The **==**, **!=**, **=~**, and **!~** operators compare their arguments as strings; all others operate on numbers. The **=~** and **!~** operators are similar to **!=** and **==**, except that the right-most side is a *pattern* against which the left-hand operand is matched. This reduces the need for use of the **switch** statement in shell procedures when all that is really needed is pattern matching.

Strings which begin with 0 are considered octal numbers. Null or missing arguments are considered 0. The result of all expressions are strings, which represent decimal numbers. It is important to note that now two components of an expression can appear in the same word; except when next to components of expressions which are syntactically significant to the parser (**&** **|** **<** **>** **(** **)**), expression components should be surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in **{** and **}** and file inquiries of the form **-l name** where *l* is one of:

```
r      Read access
w      Write access
x      Execute access
e      Existence
o      Ownership
z      Zero size
f      Plain file
d      Directory
```

The specified name is command and file-name expanded and then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible, then all inquiries return false, that is, 0. (Command runs succeed, returning true (1), if the command exits with status 0; otherwise they fail, returning false (0).) If more detailed status information is required, run the command outside an expression and the examine **status** shell variable.

Nonbuilt-in Command Execution

When a command to run is found not to be a built-in command, the shell attempts to run the command with *execve*. (See the **exec** system call in *AIX Operating System Technical Reference*.) Each word in the **path** shell variable names a directory from which the shell attempts to run the command. If it is given neither a **-c** nor a **-t** flag, the shell will hash the names in these directories into an internal table so it only tries an **exec** in a directory if there is a possibility that the command resides there. If this mechanism has been turned off with **unhash**, or if the shell is given a **-c** or **-t** (and in any case for each directory component of **path** that does not begin with a **/**), the shell concatenates with the given command name to form a path name of a file, which it then attempts to run.

Parenthesized commands always run in a subshell. Thus, `(cd ; pwd) ; pwd` displays the **home** directory without changing the current directory location, whereas `cd ; pwd` changes the current directory location to the **home** directory. Parenthesized commands are most often used to prevent **chdir** from affecting the current shell.

If the file has execute permissions, but is not an executable binary to the system, then it is assumed to be a file containing shell commands and a new shell runs to read it.

If there is an alias for shell, then the words of the alias will be prefixed to the argument list to form the shell command. The first word of the alias should be the full path name of the shell. Note that this is a special, late-occurring case of alias substitution and only allows words to be prefixed to the argument list without modification.

Signal Handling

The shell normally ignores **QUIT** signals. Jobs running detached are immune to signals generated from the keyboard (**INTERRUPT**, **QUIT**, and **HANGUP**). Other signals have the values the shell inherited from its parent. You can control the shell's handling of **INTERRUPT** and **TERMINATE** signals in shell procedures with **onintr**. Login shells catch the **TERMINATE** signal; otherwise, this signal is passed on to children from the state in the shell's parent. In no case are **INTERRUPT**s allowed when a login shell is reading the **.logout** file.

Limitations

The following are **csb** limitations:

- Words can be no longer than 1024 characters.
- Argument lists are limited to 5120 characters.
- The number of arguments to a command that involves file-name expansion is limited to 1/6th the number of characters allowed in an argument list.

- Command substitutions can substitute no more characters than are allowed in an argument list.
- To detect looping, the shell restricts the number of alias substitutions on a single line to 20.

Flags

If the first argument to the shell is - (minus), this is a login shell. The flags are interpreted as follows:

- c Reads commands from the (single) following argument, which must be present. Any remaining arguments are placed in **argv**.
- e Exits if any invoked command ends abnormally or yields a nonzero exit status.
- f Starts without searching for or running commands from the **.cshrc** file in the your home directory.
- i Prompts for its top-level input (an interactive shell), even if input does not appear to be coming from a work station. Shells are interactive without this flag if their input and output are attached to work stations.
- n Parses commands but does not run them. This aids in syntactic checking of shell procedures.
- s Takes command input from the standard input.
- t Reads and processes a single line of input. You can use a \ to escape the new-line character at the end of the current line to continue onto another line.
- v Sets the **verbose** shell variable, with the effect that command input is echoed after history substitution.
- V Sets the **verbose** shell variable even before **.cshrc** runs.
- x Sets the **echo** shell variable, so that commands are echoed immediately before they run.
- X Sets the **echo** shell variable even before **.cshrc** runs.

After processing of flag arguments, if arguments remain but none of the -c, -i, -s, or -t flags were given, the first parameter is taken as the name of a file of commands (shell procedure). The system opens this file and saves its name for possible resubstitution by \$0. If the first characters of the shell procedure are **#!/shell_pathname**, **cs** runs the specified shell to process the procedure. Otherwise, **cs** runs the standard shell (**sh**). Remaining parameters initialize the **argv** variable. For more information on the **#!/shell_pathname** comment, see the **exec** system call in *AIX Operating System Technical Reference*.

Files

\$HOME/.cshrc	Read at beginning of execution by each shell.
\$HOME/.login	Read by login shell, after .cshrc at login.
\$HOME/.logout	Read by login shell, at logoff.
/bin/sh	Standard shell.
/tmp/sh*	Temporary file for < <.
/etc/passwd	Source of home directories for ~ <i>name</i> .

Related Information

The following commands: “**cd**” on page 150, “**make**” on page 625, “**pr**” on page 761, and “**sh**” on page 913.

The **access**, **exec**, **fork**, **pipe**, **umask**, and **wait** system calls, the **a.out** and **environ** files, and the **environment** miscellaneous facility in *AIX Operating System Technical Reference*.

“Overview of International Character Support” in *Managing the AIX Operating System*.

The discussion of Japanese Language Support in *Japanese Language Support User's Guide*.

csplit

csplit

Purpose

Splits files by context.

Syntax



OL805177

Description

The **csplit** command reads a *file* and separates it into segments defined by the specified parameters (*parm . . .*). By default, **csplit** writes these segments to files **xx00** + . . . **xxn**, where *n* is the number of *parms* listed on the command line (*n* may not be greater than 99). These new files get the following pieces of *file*:

- 00: From the start of *file* up to, but not including, the line referenced by the first *parm*.
- 01: From the line referenced by the first *parm* up to the line referenced by the second *parm*.
- .
- .
- .
- n* + 1: From the line referenced by the last *parm* to the end of *file*.

Note that **csplit** does not alter the original *file*.

The specified *parms* can be a combination of the following:

/pattern/ Creates a file that contains the segment from the current line up to (but not including) the line containing *pattern*, which becomes the current line.

%pattern% Makes the line containing *pattern* the current line, but does not create a file for the segment.

+ *num*
- *num*

Moves forward or backward the specified number of lines from the line matched by an immediately preceding *pattern* parameter (for example, */Page/-5*).

- linenum* Creates a file containing the segment from the current line up to (but not including) *linenum*, which becomes the current line.
- {*number*} Repeats the preceding argument the specified *number* of times. This number can follow any of the *pattern* or *linenum* parameters. If it follows a *pattern* parameter, **csplit** reuses that *pattern* the specified number of times. If it follows a *linenum* parameter, **csplit** splits the file from that point every *linenum* of lines for the specified number of times.

Quote all *pattern* parameters that contain blanks or other characters special to the shell. Patterns may not contain embedded new-line characters. In an expression such as **[a-z]**, the minus means “through” according to the current collating sequence. A collating sequence may define *equivalence classes* for use in character ranges. See “Overview of International Character Support” in *Managing the AIX Operating System* for more information on collating sequences and equivalence classes.

Flags

- f prefix** Specifies the *prefix* name for the created file segments. **xx** is the default *prefix*.
- k** Leaves created file segments intact in the event of an error.
- s** Suppresses the display of character counts.

Examples

1. To split the text of a book into a separate file for each chapter:

```
csplit book "/^ Chapter *[0-9]/" {9}
```

This creates files named **xx00**, **xx01**, **xx02**, . . . ,**xx9**, which contain individual chapters of the file **book**. Each chapter begins with a line that contains only the word **Chapter** and the chapter number. The file **xx00** contains the front matter that comes before the first chapter. The **{9}** after the *pattern* allows up to nine chapters.

2. To specify the prefix for the created file names:

```
csplit -f chap book "/^ Chapter *[0-9]/" {9}
```

This splits **book** into files named **chap00**, **chap01**, **chap02**, . . . ,**chap9**.

Related Information

The following commands: “**ed**” on page 371, “**sh**” on page 913, and “**regcmp**” on page 820.

The **regxp** file in *AIX Operating System Technical Reference*.

“Overview of International Character Support” in *Managing the AIX Operating System*.

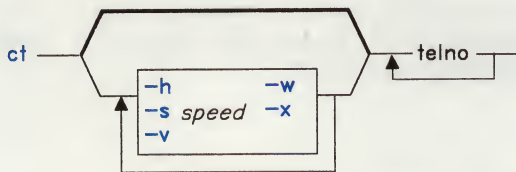
ct

ct

Purpose

Dials an attached terminal and issues a login process.

Syntax



AJ2FL101

Description

The Basic Networking Utilities (BNU) command **ct** enables a user on a remote ASCII terminal, such as an IBM 3161 or a DEC VT100, to communicate with an RT work station over a telephone line attached to a modem at each end of the connection. The user on the remote terminal can then log in and work on the RT work station.

A user on the local system issues **ct** with the appropriate telephone number to call the modem attached to the remote terminal. When the connection is established, **ct** issues an AIX login prompt that is displayed on the remote terminal. The user on the remote terminal enters an AIX login name at the prompt, and AIX opens a new shell. The user at the remote terminal then proceeds to work on the RT PC just like a local user.

Note: In order to establish a **ct** connection, the remote user generally contacts a local user (with a regular phone call) and asks the local user to issue the command.

The **ct** command is useful in the following situations:

- When a user working off site needs to communicate with a local system under strictly supervised conditions. Because the local system contacts the remote terminal, the remote user does not need to know the phone number of the local system.
- When the cost of the connection should be charged either to the local site, or to a specific account on the calling RT. If the remote user has the appropriate access permission and can make outgoing calls on the attached modem, that user can make the equivalent of a collect call. The remote user calls the specified local system, logs in, and issues the phone number of the remote terminal *without* the **-h** flag. The local

system hangs up the initial link so that the remote terminal is free for an incoming call, and then calls back to the modem attached to the remote terminal.

Note: Before issuing the **ct** command, be certain that the remote terminal is attached to a modem that can answer the telephone.

The **ct** command is not as flexible as the BNU command **cu**. For example, the user can not issue AIX commands on the local system while connected to a remote system via **ct**. However, the **ct** command does have two features not available with **cu**:

- The user can instruct **ct** to continue dialing the specified number until the connection is established or a set amount of time has elapsed.
- The user can specify more than one telephone number at a time to instruct **ct** to continue dialing each modem until a connection is established over one of the lines.

If the user specifies alternate dialing paths by entering more than one number on the command line, **ct** tries each line listed in the file **/usr/adm/uucp/Devices** until it finds an available line with appropriate attributes, or runs out of entries. If there are no free lines, **ct** asks if it should wait for one, and if so, for how many minutes. The **ct** command continues to try to open the dialers at 1-minute intervals until the specified time is exceeded. The user can override this prompt by specifying a time with the **-wn** flag when entering the command.

After the user logs off, **ct** prompts the user on the remote terminal with a reconnect option; the system can either display a new login prompt or drop the line.

Flags

- wn** Allows the dialogue to be overridden by specifying *n* as the maximum number of minutes that **ct** is to wait for a line. The command then dials the remote modem at 1-minute intervals until the connection is established or the specified time has elapsed.
- xn** Used for debugging. Produces detailed information about the command's execution on standard error output on the local system. The debugging level, *n*, is a single digit between 0 and 9. The recommended default is 9.
- h** Prevents **ct** from hanging up the current line to answer an incoming call.
- v** Allows **ct** to send a running narrative to standard error output.
- sspeed** Sets the data rate where *speed* is expressed in baud. The default is 1200.
- telno** Specifies the telephone number of the modem attached to the remote terminal. The *telno* may include the digits 0 - 9, minus signs (-) representing delays, equal signs (=) representing secondary dial tones, asterisks (*), and pound/number signs (#). The phone number may contain a maximum of 31 characters.

Examples

1. To connect to a modem with an internal number 4-1589 (the - is optional):

```
ct 41589
```

The system responds:

```
Allocated dialer at 1200 baud  
Confirm hang_up? (y to hang_up)
```

2. To dial a modem connected to a local telephone number (dialing 9 for an outside line and specifying a 3-minute wait time):

```
ct -w3 9=2453017
```

3. To dial a long-distance number (specifying an outside line and a 5-minute wait):

```
ct -w5 9=15026647003
```

Files

/usr/adm/uucp/Devices	Information about available devices.
/usr/adm/uucp/Dialcodes	Dialing code abbreviations.
/usr/adm/uucp/Dialers	Initial handshaking on a link.
/usr/adm/uucp/Permissions	Access permission codes.
/usr/adm/uucp/Systems	Accessible remote systems.

Related Information

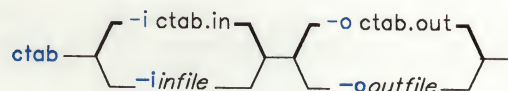
The following commands: “cu” on page 263 and “login” on page 584.

ctab

Purpose

Produces a collating table.

Syntax



OL805451

Description

The **ctab** command takes an input file (by default a file named **ctab.in** found in the current directory) and produces a binary file (by default named **ctab.out**) containing a collating table. These output files should be stored in a special directory such as **/usr/lib/nls**.

Programs that need the current collating information use the **NLCTAB** environment variable to access that information.

The following conventions are used to make it easier to set up a table file:

- One line of information is present for each character explicitly named.
- A line beginning with the word **option** serves to change one or more of the default conditions or metacharacters built into **ctab**. An **option** line contains a set of name/value pairs, with each half of each pair delimited by tab or space characters. The following is a list of recognized names:

eclass	Turns the use of equivalence classes on or off globally. The assigned value must be on (the default) or off . (This name is ignored when Japanese Language Support is installed.)
sep	Uses the assigned value as the field separator character. The default value is : (colon).
trans	Uses the assigned value of the “translate” indicator in subject character fields. The default character is (vertical bar).
repeat	Uses the assigned value as the “same as last line” indicator in subject character field. The default value is ^ (circumflex).
comment	Uses the assigned value as the comment character. The default value is the # character.

- The order of the per-character input lines specifies the collating sequence.
- By default, fields on a line are separated by colons. Tabs or spaces may surround fields or separators. You can change the separator character with an option line.
- Use an octal escape sequence in the ASCII range to name a nonprintable character. A backslash character that does not form part of a valid escape sequence serves to strip the following character, including a second backslash, of any special meaning it otherwise would have. For example, to include the colon character in the collating sequence, use the following line:

```
\::
```

The input file format includes a comment convention, namely that the remainder of the line following a # character is ignored. The comment character can be changed with an **option** line.

Input File Specification

Use the following rules to build *infile*, entering field information for each line:

1. The first field on a line contains the **subject character**, a character to be inserted into the collating sequence at that point.
 - This subject character definition can include a **translation mechanism**:
 - Instead of a single character, this field may contain two or more characters that are to be collated as a single unit, or
 - The single subject character may be followed by a vertical bar (|) and a single- or multiple-character string. The vertical bar indicates that the first character will be translated to the second string before being collated.

For example, to treat an “é” (e acute) as equivalent to the character “e,” use the following line:

```
é|e
```
 - One restriction is placed on the translation mechanism: the subject character cannot be contained in the translated string of characters. For example, the following line is illegal:

```
o|oe
```
- Any form of the first field may contain a trailing circumflex (^) to indicate that the current character is to collate to the same value as the preceding one. However, a circumflex following a translation string is illegal because the subject character to be translated has no inherent collating value.

- If the subject field contains a string of multiple characters (to collate as a unit), its first character must be declared elsewhere to establish the default collating sequence of that character.
 - The translate and collating no-change characters can be changed with **option** lines.
2. The second and third fields specify whether or not a character is alphabetic and what its lower- and upper-case equivalents are:
 - If a subject character is to be treated as a lowercase alphabetic, the second field on its line is its uppercase equivalent, and the third field must be **l** or **L**.
 - If a subject character is to be treated as an uppercase alphabetic, the second field on its line is its lowercase equivalent, and the third field must be **u** or **U**.
 - If a subject character is to be treated as a control character or a space character, the third field must be **c**, **C**, **s**, or **S**.
 - Each character explicitly named whose line contains a non-null second field will be considered alphabetic (that is, matched by **NCisalpha**). Characters that do not have an uppercase or lowercase equivalent (that is, that have a null second field) but that you wish to be considered alphabetic should simply contain a third field that is **l**, **L**, **u**, or **u**.
 3. The fourth field on a line is used explicitly to specify the first character in the **equivalence class** of the subject character. The members of one equivalence class must be consecutively listed in the input file.
 - There cannot be any gaps within a particular equivalence class. For example, the following lines will put the characters **a**, **b**, and **c** in the same equivalence class:


```
a:A:l:a
b:B:l:a
c:C:l:a
```
 - As a convenience, if the fourth field is not specified, then the group of consecutive characters with blank fourth fields, provided that they are all based on the same Roman alphabetic character, will be placed in the same equivalence class. To reiterate, only characters with the same base will be placed into the same equivalence class by default. If you wish to have many characters from different bases belong to one equivalence class, as in the preceding example, the first character of the equivalence class has to be specified in the fourth field for every character specified.

ctab

- It is illegal to specify an equivalence character that comes later in the collating sequence. The fourth field can refer only to characters that have already been mentioned.
 - All *international character support* characters not based on Roman alphabetic characters by default are the sole members of their equivalence class.
-

Japanese Language Support Information

When Japanese Language Support is installed on your system, the information about the second, third, and fourth fields is irrelevant.

Characters not named in the table file that have an ordinal value (that is, a value as an **NLchar**) below the ordinal value of the lowest-valued character named are put into the collating sequence below the first character in the table file. All other characters not named in the table file are put into the collating sequence above the last character in the table file.

The standard characters for decimal and hexadecimal digits are always marked as digits (to be matched by **NCisdigit** and **NCisxdigit**). All other printable characters not marked as alphabetic are marked as punctuation.

Flags

- i *infile* Specifies the name of the input file (**ctab.in** by default).
- o *outfile* Specifies the name of the output file (**ctab.out** by default).

Files

- /usr/lib/nls/ascii.ctab Input file listing the ASCII range of characters.
- /usr/lib/nls/example.ctab Input file listing a sample Collating Sequence.

Related Information

The **NCisalpha**, **NCisdigit**, **NCisxdigit**, **nls**, and **NLgetenv** subroutines in *AIX Operating System Technical Reference*.

“Overview of International Character Support” in *IBM RT Managing the AIX Operating System*.

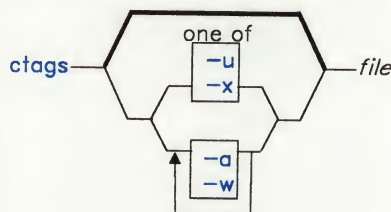
The discussion of Japanese Language Support in *Japanese Language Support User's Guide*.

ctags

Purpose

Makes a file of tags to help locate objects in source files.

Syntax



OL805457

Description

The **ctags** command makes a tags file for **ex** and **vi** editors from the specified C, Pascal, and FORTRAN source files. A tags file gives the locations of specified objects (in this case functions) in a group of files. Each line of the tags file contains the object name, the file in which it is defined, and an address specification for the object definition. Functions are searched with a pattern. Specifiers are given in separate fields on the line, separated by blanks or tabs. Using the tags file, **ex** and **vi** can quickly find these object definitions.

If a file name ends in **.c** or **.h**, it is assumed to be a C source file and is searched for C routine and macro definitions. Others are first examined to see if they contain any Pascal or FORTRAN routine definitions; if not, they are processed again for C definitions.

The tag **main** is treated specially in C programs. The tag formed is created by prefixing **M** to the file name, removing a trailing **.c** (if any), and removing the leading path name components. This makes use of **ctags** practical in directories with more than one program.

Notes:

1. Recognitions of the keywords **function**, **subroutine**, and **procedure** in FORTRAN and Pascal code is performed in a very simple-minded way. No attempt is made to deal with block structure; if you have two Pascal procedures with the same name but in different blocks, **ctags** may yield inadequate results.
2. The **ctags** command does not know about **#ifdef**.

ctags

Flags

- a Appends to tags file.
- w Suppresses warning diagnostics.
- x Causes **ctags** to display a list of object names, the line number and file name on which each is defined, as well as the text of that line. This provides a simple index. If you specify this flag, **ctags** does not build a tags file.
- u Updates the specified files in tags; that is, all references to them are deleted, and the new values are appended to the file. This flag may be slow. (It is usually faster to simply rebuild the tags file.)

Files

tags Output tags file.

Related Information

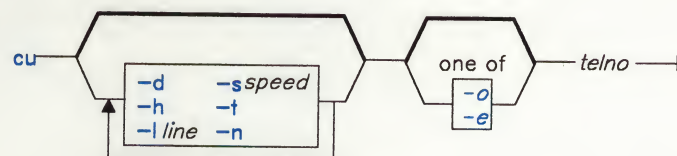
The following commands: “**ex**” on page 407 and “**vi**, **vedit**, **view**” on page 1187.

cu

Purpose

Connects directly or indirectly to another UNIX system.

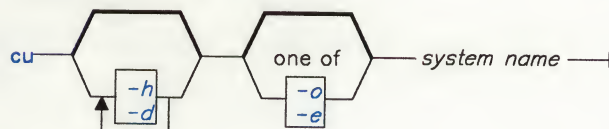
Syntax



OL805553



OL805554



OL805555

Description

The Basic Networking Utilities (BNU) command `cu` connects one system to another UNIX system, to a terminal connected to a UNIX system, or, if the proper hardware and software are installed, to a non-UNIX system. The connection can be established over a hard-wired line, or over a telephone line via a modem.

Once the connection is established, a user can be logged in on both systems at the same time, executing commands on either one without dropping the BNU communication link. If the remote computer is also running under UNIX, the user can transfer ASCII files between the two systems.

Note: The system should already be configured to use the `cu` command. Refer to *Managing the AIX Operating System* for details about this configuration.

After issuing **cu** from the local system, the user must press the **Enter** key (carriage return) and then log in to the remote system.

After making the connection, **cu** runs as two concurrent processes: the ***transmit process*** reads data from standard input and, except for lines beginning with a **~** (tilde), passes that data to the remote terminal. The ***receive process*** accepts data from the remote system and, except for lines beginning with a **~**, passes it to standard output. To control input from the remote system so the buffer is not overrun, **cu** uses an automatic DC3/DC1 (**Ctrl-Q/Ctrl-S**) protocol.

In addition to issuing regular AIX commands on the remote system, the user can also issue special **cu** "local commands," which are preceded by a **~**. Use these **~** commands to issue AIX commands on the local system and to perform tasks such as transferring files between two UNIX systems.

Local ~ Commands

The transmit process interprets lines beginning with a tilde in the following ways:

~.	Logs the user off the remote computer and terminates the remote connection.
~!	Returns the user to an interactive shell on the local system. Toggle between the local and remote systems using ~! (remote to local) and Ctrl-D (local to remote).
~!cmd...	Executes the command denoted by <i>cmd</i> on the local system via sh -c .
~\$cmd...	Runs the command denoted by <i>cmd</i> locally and sends its output to the remote system for execution.
~%cd	Changes the directory on the local system.
~%take from [to]	Copies the <i>from</i> file on the remote system to the <i>to</i> file on the local system. If <i>to</i> is omitted, the remote file is copied to the local system under the same file name. As each block of the file is transferred, consecutive single digits are displayed on the terminal screen.
~%put from [to]	Copies the <i>from</i> file on the local system to the <i>to</i> file on the remote system. If <i>to</i> is omitted, the local file is copied to the remote system under the same file name. As each block of the file is transferred, consecutive single digits are displayed on the terminal screen.
~ ~line	Sends the string denoted by <i>~line</i> to the remote system.
~%break	Transmits a BREAK to the remote system. The BREAK can also be specified as ~%b .
~%debug	Toggles the -debug flag on or off; this can also be specified as ~%d .
~t	Prints the values of the TERMIO structure variables for the user's terminal. This is useful for debugging.

- ~l** Prints the values of the **TERMIO** structure variables for the remote communication line. This is useful for debugging.
- ~%nostop** Toggles between DC3/DC1 input control protocol and no input control. This is useful in case the remote system is one that does not respond properly to the DC3 and DC1 characters.

Note: As soon as the user enters **~!**, **~%**, **~\$**, **~t**, or **~l**, the system displays the name of the local computer in a format such as the following:

```
~[system_name]!/%
```

The user then enters the command to be executed on the local computer.

Additional Information

- The receive process normally copies data from the remote system to the local system's standard output. Internally, the program accomplishes this by initiating an output diversion to a file when a line from the remote system begins with **~>**.
Data from the remote system is diverted to *file* on the local system. The trailing **~>** marks the end of the diversion.
- The use of **~%put** requires **stty** and **cat** on the remote system. It also requires that the current erase and kill characters on the remote system be identical to these current control characters on the local system. Backslashes are inserted at appropriate places.
- The use of **~%take** requires **echo** and **cat** on the remote system. Also, **stty tabs** mode should be set on the remote system if tabs are to be copied without expansion to spaces.
- The **cu** command can be used to connect multiple systems, and commands can then be executed on any of the connected systems. For example, issue **cu** on system X to connect to system Y, and then issue **cu** on system Y to connect to system Z. System X is then the local computer, and systems Y and Z are remote computers.

The user can execute commands on system Z by logging in and issuing the command. Commands can be executed on system X by prefixing the command with a single tilde (**~cmd**), and on system Y by prefixing the command with two tildes (**~~cmd**). In general, one tilde causes the specified command to be executed on the original local computer, and two tildes cause the command to be executed on the next system on which **cu** was issued.

For example, once the multiple systems are connected, the user can execute the **uname** command with the **-n** flag (to display the node name) on Z, X, and Y as follows:

```
$ uname -n
Z
$ ~!uname -n
X
$ ~~!uname -n
Y
```


Notes:

1. After executing **cu**, the user must log in to the remote system and press **Enter** (carriage return).
2. The **cu** command does not do integrity checking on data it transfers.
3. Data fields with special **cu** characters may not be transmitted properly.
4. Depending on the interconnection hardware, it may be necessary to use a ~. to terminate the conversation even if the normal logoff sequence has been used.
5. There is an artificial slowing of transmission by **cu** during the ~%**put** operation so that loss of data is unlikely.
6. The exit code is 0 for normal exit, otherwise, -1.

Flags

-sspeed Specifies the transmission speed (300, 1200, 2400, 4800, 9600). The default value is "Any" speed, which instructs the system to use the rate appropriate for the default (or specified) transmission line. (The order of the transmission lines is specified in the **/usr/adm/uucp/Devices** file.) Most modems operate at 300, 1200, or 2400 baud, while most hard-wired lines are set to 1200 baud or higher.

-lline Specifies a device name to use as the communication line. This can be used to override the search that would otherwise take place for the first available line with the right speed. When the **-l** flag is used without the **-s** flag, the speed of a line is taken from the **/usr/adm/uucp/Devices** file.

When the **-l** and **-s** flags are used together, **cu** searches the **/usr/adm/uucp/Devices** file to check whether the requested speed is available for the specified line. If so, the connection is made at the requested speed; otherwise, an error message is printed, and the call is not made.

The specified device is generally a hard-wired asynchronous line (for example, **/dev/tty2**), in which case a telephone number (*telno*) is not required. If the specified device is associated with a modem, a telephone number must be provided. Using this flag with *system_name* rather than with *telno* does not give the desired result (see *system_name*, below).

Note: Under ordinary circumstances, the user should not have to specify the transmission speed, or a line/device. The defaults set when BNU is installed should be sufficient. Refer to *Managing the AIX Operating System* for information about setting defaults.

-h Emulates local echo, supporting calls to other systems that expect terminals to be set to half-duplex mode.

-t	Used to dial an ASCII terminal that has been set to auto answer. Appropriate mapping of carriage-return to carriage-return line-feed pairs is set.
-d	Prints diagnostic traces.
-o	Designates that odd parity is to be generated for data sent to the remote system.
-e	Designates that even parity is to be generated for data sent to the remote system.
-n	For added security, prompts the user to provide the telephone number to be dialed, rather than taking it from the command line.
<i>telno</i>	When using a modem, the argument is the telephone number, with appropriately placed equal signs for secondary dial tones, or minus signs for delays of 4 seconds.
<i>system_name</i>	A uucp system name can be used rather than a telephone number; in that case, cu obtains an appropriate hard-wired line or telephone number from /usr/adm/uucp/Systems . System names must be ASCII characters only. Note: Do not use the <i>system_name</i> flag in conjunction with the -l and -s flags. If you do, cu connects to the first available line for the requested system name, ignoring the specified line and speed.

Examples

1. To connect to a remote system using a system name:
`cu hera`
2. To dial a remote system whose telephone number is 1-201-555-1212, where dialing 9 is required to get an outside dial tone and the baud rate is 1200:
`cu -s 1200 9=12015551212`
If the speed is not specified, "Any" is the default value.
3. To log in to a system connected by a hard-wired line:
`cu -l /dev/tty2`
or
`cu -l tty2`
4. To dial a remote system with the specified line and a specific speed:
`cu -s 1200 -l tty3`

cu

5. To dial a remote system using a specific line associated with a modem:

```
cu -l cu14 9=12015551212
```

6. To copy a file from the local system to the remote system (after logging in to the remote system):

```
~%put /u/amy/file
```

or

```
~%put /u/amy/file /u/amy/tmpfile
```

Files

/etc/locks/LCK..(tty-device)

Prevents multiple use of device.

/usr/adm/uucp/Devices

Information about available links.

/usr/adm/uucp/Dialcodes

Dialing code abbreviations.

/usr/adm/uucp/Dialers

Initial handshaking on a link.

/usr/adm/uucp/Permissions

Access permission codes.

/usr/adm/uucp/Systems

Accessible remote systems.

Related Information

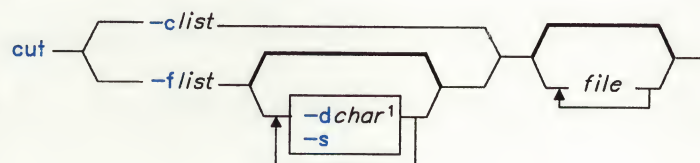
The following commands: “**cat**” on page 137, “**ct**” on page 254, “**echo**” on page 369, “**stty**” on page 1018, “**uname**” on page 1151, and “**uucp**” on page 1144.

cut

Purpose

Writes out selected fields from each line of a file.

Syntax



¹ The default *char* is a tab.

OL805178

Description

The **cut** command cuts out columns from a table or fields from each line of a file and writes these columns or fields to standard output. If you do not specify a *file*, **cut** reads standard input.

You must specify either the **-c** or **-f** flag. The *list* parameter is a comma-separated and/or minus-separated list of integer field numbers (in increasing order). The minus separator indicates ranges. Some sample *lists* are 1,4,7; 1-3,8; -5,10 (short for 1-5,10); and 3- (short for third through last field). The fields specified by *list* can be a fixed number of character positions, or the length can vary from line to line and be marked with a field delimiter character, such as a tab character.

You can also use the **grep** command to make horizontal cuts through a file and the **paste** command to put the files back together. To change the order of columns in a file use **cut** and **paste**.

Flags

- c list** Specifies character positions. For example, if you specify **-c1-72**, **cut** writes out the first 72 characters in each line of the file. Note that there is no space between **-c** and *list*.

cut

-dchar Uses the specified *character* as the field delimiter when you specify the **-f** flag. You must quote characters with special meaning to the shell, such as the space character. Any ASCII character can be used as *char*.

Japanese Language Support Information

char can either be any ASCII character, or any SJIS character.

-flist Specifies a list of fields assumed to be separated in the file by a delimiter character, by default the tab character. For example, if you specify **-f1,7**, **cut** writes out only the first and seventh fields of each line. If a line contains no field delimiters, **cut** passes them through intact (useful for table subheadings), unless you specify the **-s** flag.

-s Suppresses lines that do not contain delimiter characters (use only with the **-f** flag).

Example

To display several fields of each line of a file:

```
cut -f1,5 -d: /etc/passwd
```

This displays the login name and full user name fields of the system password file. These are the first and fifth fields (**-f1,5**) separated by colons (**-d:**).

So, if the **/etc/passwd** file looks like this:

```
su:UHuj9Pgdvz0J":0:0:User with special privileges::/bin/sh
daemon*:1:1::/etc:
bin*:2:2::/bin:
sys*:3:3::/usr/src:
adm*:4:4:System Administrator:/usr/adm:/bin/sh
pierre:boodwqT3irHFE:200:200:Pierre Harper:/u/pierre:/bin/sh
joan:wijBNaYpCZuL.:202:200:Joan Brown:/u/joan:/bin/sh
```

then **cut** produces:

```
su:User with special privileges
daemon:
bin:
sys:
adm:System Administrator
pierre:Pierre Harper
joan:Joan Brown
```

Related Information

The following commands: “**grep**” on page 501 and “**paste**” on page 736.

The discussion of Japanese Language Support in *Japanese Language Support User's Guide*.

cvid

cvid

Purpose

Creates a VRM install diskette for backup purposes.

Syntax

`cvid_ device_ -f fs-id -v vol-id prototypefile`

OL805104

Description

The **cvid** command backs up the VRM minidisk onto a diskette. Since you can reinstall the VRM system from this backup diskette, use **cvid** as a precautionary measure before modifying the VRM. You must be a member of the system group or operating with superuser authority to run this command.

The *device* parameter specifies the device (special file) to which **cvid** copies the VRM. This can be a block device name or a directory name. If *device* is a directory name, **cvid** reads the `/etc/filesystems` file for the corresponding device. **cvid** uses the *prototypefile* parameter to determine the size of the new file system. *prototypefile* defaults to `/vrmm/vproto`. For more information on prototype files, see “**mkfs**” on page 658 and “**proto**” on page 780.

When auditing is on, an audit record of the type **cvid** is created.

Warning: If you used the **mv** command to change or modify the order of the files on a minidisk, you could lose files when restoring a **cvid** backup of the minidisk. To avoid problems, modifications, additions or changes to the `/vrmm/ldlist/posts` directory must be reflected in the `/vrmm/inst.batch` file.

Flags

- `-f fs-ID` Makes *fs-ID* the label for the new file system. The default label is **vrmmnt**.
- `-v vol-ID` Makes *vol-ID* the volume label for the new file system. The default label is **ibmvrmm**.

Files

etc/filesystems
/vrm/proto

Contains device directories.
Contains the default file size for new file systems.

Related Information

The following commands: “**mkfs**” on page 658 and “**mount**” on page 669.

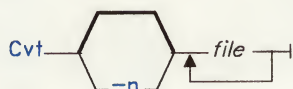
Cvt

Cvt

Purpose

Moves old UUCP files into new BNU directories.

Syntax



AJ2FL122

Description

The **Cvt** command moves existing (old) UNIX-to-UNIX Copy Program (UUCP) data and command files into new Basic Networking Utilities (BNU) directories.

After the new BNU programs are installed, issue **Cvt** before attempting to use the BNU functions. The **Cvt** shell handles the command (**C.***) and data (**D.***) files created under the old UUCP facility so that they will run under the new BNU facility. The command first creates the required new BNU directories and then moves the old UUCP files (located in the **/usr/spool/uucp** directory) into those directories.

To issue **Cvt** from the command line, a user must have superuser privileges.

Note: If **Cvt** is not used to move old UUCP command and data files into new BNU directories, those files will not run after the new BNU Program is installed.

Flags

- n** Displays the message explaining the actions that **Cvt** performs but does not execute the command. A *file* can be specified for **D.*** and **C.*** files, but not for **X.*** (execute) files.

Files

/usr/adm/uucp
/usr/spool/uucp

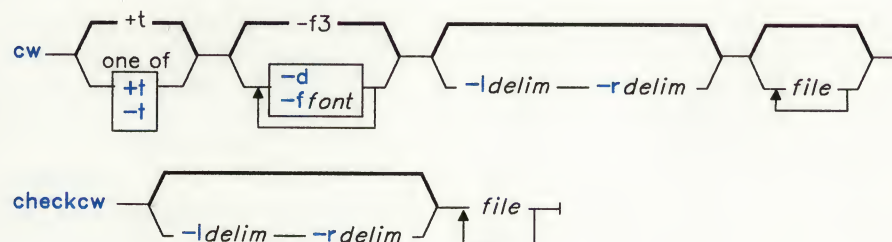
Directory in which **Cvt** is stored.
Spooling directory.

cw, checkcw

Purpose

Prepares constant-width text for **troff**.

Syntax



OL805427

Description

The **cw** command preprocesses **troff** files containing text to be typeset in the constant-width (CW) font. **cw** reads standard input if you do not specify a *file* or if you specify a -. (minus) as one of the input file names. It writes its output to standard output.

Since the text that is typeset by **cw** resembles the output of line printers and work stations, it can be used to typeset examples of programs and computer output in user manuals and programming texts. It has been designed to be distinctive when used with the Times Roman font.

Because the CW font contains a “nonstandard” set of characters and because text typeset with it requires different character and interword spacing than is used for “standard fonts,” you must use **cw** to preprocess documents that use the CW font.

The CW font contains the 94 printing ASCII characters:

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
!$%&()'"+@.,/:;=?[]|_~" '<>{}#\
```

plus eight non-ASCII characters represented by four-character **troff** strings (in some cases attaching these strings to "nonstandard" graphics):

Character	Symbol	Troff Name
"Cents" sign	¢	\(ct
EBCDIC "not" sign	¬	\(no
Left arrow	←	\(<-
Right arrow	→	\(>-)
Down arrow	↓	\(da
Vertical single quote	'	\(fm
Control-shift sign	␣	\(dg
Visible space sign	□	\(sq
Hyphen	—	\(hy
Up arrow	↑	\(ua
Home arrow	↖	\(lh

OL805409

The **cw** command recognizes five request lines, as well as user-defined delimiters. The request lines look like **troff** macro requests. **cw** copies them in their entirety onto the output. Thus, you can define them as **troff** macros; in fact, the **.CW** and **.CN** macros *should* be so defined. The five requests are:

- .CW** Marks the start of text to be set in the CW font. This request causes a break. It can take the same flags (in the same format) as those available on the **cw** command line.
- .CN** Marks the end of text to be set in the CW font. This request causes a break. It can take the same flags (in the same format) as those available on the **cw** command line.
- .CD** Changes the delimiters and/or settings of other flags. It can take the same flags (in the same format) as those available on the **cw** command line. The purpose of this request is to allow the changing of flags other than at the beginning of a document.
- .CP** *argument-list* Concatenates all the arguments (delimited like **troff** macro arguments), with the odd-numbered arguments set in the CW font and the even-numbered ones in the prevailing font.
- .PC** *argument-list* Acts the same as **.CP**, except the even-numbered (rather than odd-numbered) arguments are set in CW font.

The **.CW** and **.CN** requests should bracket text that is to be typeset in the CW font “as is.” Normally, **cw** operates in the transparent mode. In that mode, every character between **.CW** and **.CN** request lines represents itself, except for the **.CD** request and the special four-character names listed previously. In particular, **cw** arranges for all periods (.) and apostrophes (') at the beginning of lines, and all backslashes (\) and ligatures (fi, ff, and so on) to be hidden from **troff**. The transparent mode can be turned off by using the **-t** flag, in which case normal **troff** rules apply. In either case, **cw** hides from the user the effect of the font changes generated by the **.CW** and **.CN** requests.

You can also use the **-l** and **-r** flags to define delimiters with the same function as the **.CW** and **.CN** requests. They are meant to enclose words or phrases that are to be set in CW font in the running text. **cw** treats text between delimiters as it does text bracketed by **.CW/.CN** pairs, with one exception. Spaces within **.CW/.CN** pairs have the same width as other CW characters, while spaces within delimited text are half as wide, so they have the same width as spaces in the prevailing text. Delimiters have no special meaning inside **.CW/.CN** pairs.

The **checkcw** command checks that left and right delimiters, and the **.CW/.CN** pairs are properly balanced. It prints out all lines in the section with the unmatched delimiters.

Notes:

1. It is unwise to use . (period) or \ (backslash) as delimiter characters.
2. Certain CW characters do not combine well with certain Times Roman characters; for example, the spacing between a CW & (ampersand) followed by a Times Roman comma (,). In such cases, using **troff** half-and quarter-space requests can help.
3. The **troff** code produced by **cw** is difficult to read.
4. The **mm** and **mv** macro packages contain definitions of **.CW** and **.CN** macros that are adequate for most use. If you define your own, make sure that the **.CW** macro invokes the **troff** no-fill (**.nf**) mode, and the **.CN** macro restores the fill mode (**.fi**), if appropriate.
5. When set in running text, the CW font is meant to be set in the same point size as the rest of the text. In displayed matter, on the other hand, it can often be profitably set one point smaller than the prevailing point size. The CW font is sized so that, when it is set in 9-point, there are 12 characters per inch.
6. Documents that contain CW text may also contain tables and equations. If this is the case, the order of preprocessing must be **cw**, **tbl**, and **eqn**. Usually, the tables will not contain any CW text, although it is possible to have elements in the table set in the CW font. Care must be taken that **cw** does not modify the **tbl** format information. Attempts to set equations in the CW font are not likely to be pleasing or successful.
7. In the CW font, overstriking is most easily accomplished with backspaces. Because spaces (and therefore backspaces) are half as wide between delimiters as inside **.CW/.CN** pairs, two backspaces are required for each overstrike between delimiters.

Flags

- d** Displays the current flag settings on the standard error output in the form of **troff** comment lines. This flag is meant for debugging.
- ffont** Replaces *font* with the **cw** font (default=3, replacing the bold font). **-f5** is commonly used for formatters that allow more than four simultaneous fonts.
This flag is useful only on the command line.
- ldelim** Sets the left delimiter as the one-or two-character string *delim*. The left delimiter is undefined by default.
- rdelim** Set the right delimiter as *delim* The right delimiter is undefined by default. The left and right delimiters may (but need not) be different.
- t** Turns the transparent mode *off*.
- +t** Turns the transparent mode *on* (this is the default).

Files

/usr/lib/font/ftCW CW font-width table.

Related Information

The following commands: “**eqn**, **neqn**, **checkeq**” on page 395, “**mmt**, **checkmm**” on page 666, “**tbl**” on page 1053, and “**troff**” on page 710.

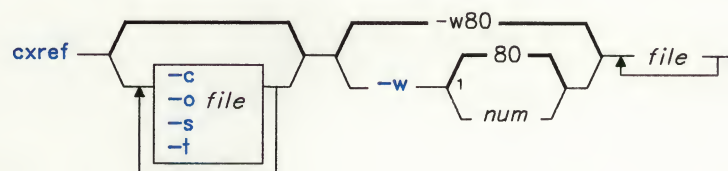
The **mm** and **mv** miscellaneous facilities in *AIX Operating System Technical Reference*.

cxref

Purpose

Creates a C program cross-reference listing.

Syntax



¹ Do not put a space between these items.

OL805180

Description

The **cxref** command analyzes C program *files* and creates a cross-reference table, using a version of the **cpp** command to include **#define** directives in its symbol table. It writes to standard output a listing of all symbols in each file processed, either separately or in combination (see the **-c** flag on page 279). When a reference to a symbol is that symbol's declaration, an * (asterisk) precedes it.

You can also use the **-D**, **-I**, and **-U** flags from the **cpp** command.

Flags

- c** displays a combined listing of the cross-references in all input files.
- o file** Directs the output to the specified *file*.
- s** Does not display the input file names.
- t** Makes the listing 80 columns wide.
- w[num]** Makes the listing *num* columns wide, where *num* is a decimal integer greater than or equal to 51. If you do not specify *num* or if *num* is less than 51, the listing will be 80 columns wide.

cxref

File

/usr/lib/xcpp Special version of C-preprocessor.

Related Information

The following commands: “**cc**” on page 140 and “**cpp**” on page 210.

The discussion of **cxref** in *AIX Operating System Programming Tools and Interfaces*.